# Towards Usability in Private Data Analytics

**Thesis approved by**
**the Department of Computer Science**
**Technische Universität Kaiserslautern**
**for the award of the Doctoral Degree**
**Doctor of Engineering (Dr.-Ing.)**

**to**

Reinhard Richard Munz

| | |
|---|---|
| Date of Defense: | December 6, 2019 |
| Dean: | Prof. Dr.-Ing. Stefan Deßloch |
| Reviewer: | Prof. Dr. Paul Francis |
| Reviewer: | Prof. Dr. Deepak Garg |
| Reviewer: | Prof. Dr. Matteo Maffei |

D 386

To my wife, Lisette,
my parents, Sibylle and Heinrich,
and my brothers, Jürgen and Wilfried.

# Abstract

Private data analytics systems preferably provide required analytic accuracy to analysts and specified privacy to individuals whose data is analyzed. Devising a general system that works for a broad range of datasets and analytic scenarios has proven to be difficult.

Despite the advent of differentially private systems with proven *formal* privacy guarantees, industry still uses inferior *ad-hoc* mechanisms that provide better analytic accuracy. Differentially private mechanisms often need to add large amounts of noise to statistical results, which impairs their usability.

In my thesis I follow two approaches to improve the usability of private data analytics systems in general and differentially private systems in particular. First, I revisit ad-hoc mechanisms and explore the possibilities of systems that do not provide Differential Privacy or only a weak version thereof. Based on an attack analysis I devise a set of new protection mechanisms including *Query Based Bookkeeping* (QBB). In contrast to previous systems QBB only requires the history of analysts' queries in order to provide privacy protection. In particular, QBB does not require knowledge about the protected individuals' data.

In my second approach I use the insights gained with QBB to propose UniTraX, the first differentially private analytics system that allows to analyze part of a protected dataset without affecting the other parts and without giving up on accuracy. I show UniTraX's usability by way of multiple case studies on real-world datasets across different domains. UniTraX allows more queries than previous differentially private data analytics systems at moderate runtime overheads.

# Acknowledgements

Many people contributed to this thesis. I would like to thank everyone that helped me through the past eight years. In the following, I would like to express my gratitude to the most notable companions.

First and foremost I would like to thank my adviser, Paul Francis. He gave me the freedom to shape this thesis and introduce and realize my very own ideas with it. He motivated me to create private analytics systems that are usable first and private second. He guided me through the longer part of my PhD time at MPI-SWS and was patient with me during those times that made it the longer part.

I would like to thank my other committee members, Deepak Garg and Matteo Maffei, for their feedback and help. Their criticism and hard questions helped me improve my research in many ways. I am grateful to them and Fabienne Eigner for their help on UniTraX. It was their expertise on formal models and proofs that turned UniTraX into a provably private system.

Unfortunately, I never had the opportunity to collaborate with any members of my research group except my adviser. Nevertheless, I would like to thank Imran Khan and İstemi Ekin Akkuş for the time we spent together. Kudos to Ekin and Ruichuan Chen, another former member of the group, for being such great hosts during my internship at Nokia Bell Labs.

Special thanks go to all other former and present members of MPI-SWS. In particular, I would like to mention Umut Açar and Allen Clement, my advisers during the first years. In their research groups I had the pleasure to work with Arthur Charguéraud, Ezgi Çiçek, Matthew Hammer, Mike Rainey, Mustafa Zengin, Nancy Estrada, and Natacha Crooks.

From the technical and support staff that work behind the scenes I would like to especially thank Carina Schmitt, Christian Klein, Claudia Richter, Maria-Louise Albrecht, Rose Hoberman, and Vera Schreiber. Whenever I had any problem it was mostly them who had to deal with me and my usually special requests. Thanks for your patience and care.

I found many friends in the institute. I would like to thank Anjo Vahldiek-Oberwagner, Aastha Mehta, Arpan Gujarati, Felipe Cerqueira, Juhi Kulshrestha, Mainack Mondal, Manohar Vanga, Oana Goga, Paarijaat Aditya, Przemyslaw Grabowicz, and Raul Herbster for making the few time besides work worthwhile.

Apart from the mentioned I had the joy to interact with many more people in and around the institute. In the interest of space I only noted those whom I spent most time with or caused most troubles for. However, these are non-exhaustive samples and I would like to thank everyone else as well. It has been an awesome time.

At this point I would like to extend these acknowledgements and thank people outside MPI-SWS. Special thanks go to my friends from CMU, Uni Würzburg, and my very close friends from high school, church community, and the summer camp instructor team. Thanks for keeping me posted and invited although I am rarely able to be part of any activities.

The most important people, however, come at the end. Without them I would have given up long before. I am grateful to my family, my parents, Sibylle and Heinrich, and my brothers, Jürgen and Wilfried. Without their love and moral support I would not have made it.

Finally, I am most thankful to Lisette. It has been her never ending love and care that kept me going through these years.

# Publications

Parts of this thesis have appeared in the following publications.

1. Reinhard Munz, Fabienne Eigner, Matteo Maffei, Paul Francis, and Deepak Garg. "UniTraX: Protecting Data Privacy with Discoverable Biases". In: *Proceedings of the 7th International Conference on Principles of Security and Trust (POST'18).* Edited by Lujo Bauer and Ralf Küsters. Volume 10804. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2018, pages 278–299. DOI: 10.1007/978-3-319-89722-6_12

1'. Reinhard Munz, Fabienne Eigner, Matteo Maffei, Paul Francis, and Deepak Garg. *UniTraX: Protecting Data Privacy with Discoverable Biases.* Technical report MPI-SWS-2018-001. Kaiserslautern and Saarbrücken: Max Planck Institute for Software Systems (MPI-SWS), Feb. 2018. URL: https://www.mpi-sws.org/tr/2018-001.pdf

2. Paul Francis, Sebastian Probst Eide, and Reinhard Munz. "Diffix: High-Utility Database Anonymization". In: *Proceedings of the 5th Annual Privacy Forum (APF'17).* Edited by Erich Schweighofer, Herbert Leitold, Andreas Mitrakas, and Kai Rannenberg. Volume 10518. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2017, pages 141–158. DOI: 10.1007/978-3-319-67280-9_8

2'. Paul Francis, Sebastian Probst Eide, Pawel Obrok, Cristian Berneanu, Sasa Juric, and Reinhard Munz. *Extended Diffix.* 2018. arXiv: 1806.02075 [cs.CR]

The following publications are not part of this thesis and vice versa.

3. Iulian Moraru, David G. Andersen, Michael Kaminsky, Nathan Binkert, Niraj Tolia, Reinhard Munz, and Parthasarathy Ranganathan. *Persistent, Protected and Cached: Building Blocks for Main Memory Data Stores.* Technical report CMU-PDL-11-114 v2. Pittsburgh: Parallel Data Laboratory (PDL), Carnegie Mellon University (CMU), Nov. 2012. URL: http://www.pdl.cmu.edu/PDL-FTP/NVM/CMU-PDL-11-114v2.pdf

3'. Iulian Moraru, David G. Andersen, Michael Kaminsky, Nathan Binkert, Niraj Tolia, Reinhard Munz, and Parthasarathy Ranganathan. *Persistent, Protected and Cached: Building Blocks for Main Memory Data Stores.* Technical report CMU-PDL-11-114. Pittsburgh: Parallel Data Laboratory (PDL), Carnegie Mellon University (CMU), Dec. 2011. URL: http://www.pdl.cmu.edu/PDL-FTP/NVM/CMU-PDL-11-114.pdf

# Vita

## Higher Education

| | |
|---|---|
| *Max Planck Institute for Software Systems (MPI-SWS)* | Jul'11–*present* |
| Doctoral student, Computer Science | |
| Enrolled at Technische Universität Kaiserslautern | Oct'14–*present* |
| *Carnegie Mellon University (CMU)* | Sep'09–Dec'10 |
| Master of Science in Information Technology with Specialization in Very Large Information Systems GPA: 3.87/4.33 | |
| *Julius-Maximilians-Universität Würzburg* | Oct'06–Sep'14 |
| Vordiplom (German Intermediate Diploma) | Apr'08 |
| Computer Science, Grade: 1 (1=excellent, 2=very good, 3=good, 4=pass, 5=fail) | |

## Research Experience

| | |
|---|---|
| *UniTraX* | Apr'16–*present* |
| Security & Privacy at MPI-SWS | |
| Developing methods to improve yield on limited query budgets without reducing result accuracy. Designing and implementing budget tracking solely based on queries, which allows to publish tracking state without the danger of leaking sensitive information.—Publication at POST'18 | Paul Francis<br>Deepak Garg<br>Matteo Maffei |
| *Towards Low Noise and Large Budgets* | May'15–*present* |
| Security & Privacy at MPI-SWS | |
| Designing systems that provide graceful deviation from privacy guarantees in order to allow more queries with less noise on results.—Submission to NDSS'17 | Paul Francis |

| | |
|---|---|
| *Giving Up on Privacy Guarantees*<br>Security & Privacy at MPI-SWS | Jun'14–*present* |
| Investigating practical alternatives to "Differential Privacy". Working towards support of unbounded numbers of queries with bounded result noise.—Publication at APF'17 | Paul Francis |
| *Alsviðr*<br>Autonomous Software Systems Research at Nokia Bell Labs | Jan'17–May'17 |
| Designed data layer for serverless computing platform. Evaluated performance of existing distributed storage systems. | Volker Hilt |
| *Triage*<br>Distributed, Dependable, and Mobile Systems at MPI-SWS | May'13–May'14 |
| Improved user satisfaction through more accurate backend response time prediction. Selective execution of requests with good prognosis raises system goodput.—Poster at SOSP'13 | Allen Clement<br>Björn Brandenburg |
| *Multi-KV*<br>Distributed, Dependable, and Mobile Systems at MPI-SWS | Nov'12–Aug'13 |
| Built mechanisms for consistency in geo-replicated storage systems.—Poster at EuroSys'13 | Allen Clement |
| *Work Stealing at Scale*<br>Programming Languages and Systems at MPI-SWS | Jan'12–Oct'12 |
| Scaled work stealing schedulers to clusters of machines. | Umut Acar |
| *PCM-KV: A Key-Value Store on Phase Change Memory*<br>Parallel Data Lab (PDL) at CMU | Sep'10–Dec'10 |
| Developed OS mechanisms to handle shortcomings of non-volatile main memory alternatives. Evaluated wear-leveling memory allocator. —Technical Report at PDL'12 | David Andersen<br>Michael Kaminsky |

## Honors & Awards

| | |
|---|---|
| *ACM SOSP Student Scholarship* | Nov'13 |
| *USENIX OSDI Student Grant* | Oct'12 |
| *Max Weber-Programm*, Study Abroad Grant | Sep'09 |
| *Max Weber-Programm*, Fellowship for gifted students | Oct'08–Mar'12 |

## Skills

German (mother tongue), English (fluent, CEFR C2)
Java, C#, C++, Python

## Extra-Curricular Activities

*"Snack Shop"* Apr'13–*present*
Personal initiative to improve the work environment by selling beers and snacks at MPI-SWS.

*"Allowio" Kolping Zeltlager Amorbach* 2002–*present*
Summer camp instructor for ∼75 children.

## Contact

Reinhard Munz
MPI-SWS
Campus E1.5
66123 Saarbrücken
Germany

munz@mpi-sws.org
+49 681 9303-9100

# Contents

# List of Figures

# 1. Introduction

Private data analytics is a long-standing and complex research problem. The goal is to provide a general data analytics interface for analysts to obtain statistical results over individuals' data without violating any single individual's privacy. To be generally applicable desired solutions are independent of specific analytic scenarios as well as specific datasets. They further conform to differing analytic accuracy requirements of analysts as well as differing privacy preferences of individuals.

Until the early 2000's providing privacy was simply to prevent an adversarial analyst from inferring any fact about a single individual. A diverse set of techniques was proposed, e.g., data swapping, rounding, or value aggregation [1, 3, 4, 28, 35]. Unfortunately, there was also a diverse set of attacks shown [21, 22, 31, 33, 34, 64, 98, 99]. These attacks render proposed defenses void, except for specific hypothetical scenarios. As a result, none of the proposed protective techniques is able to provide a meaningful amount of privacy protection.

In 2002 Sweeney introduced $k$-anonymity [109, 110], the first data analytics technique to provide a configurable privacy parameter, $k$. Individuals are guaranteed to be indistinguishable to a group of at least $k-1$ other individuals. To provide indistinguishability the original data must be modified and individuals' data either be changed or removed. These modifications introduce non-uniform noise that is highly dependent on the particular implementation. The amount of noise and how it effects specific analytic queries cannot be communicated to analysts. Therefore, $k$-anonymity is in general unable to guarantee analytic accuracy. To improve accuracy the original paper suggests to selectively apply $k$-anonymity only to parts of the data. Partial application, however, leads to partial protection and subsequently requires additional mitigation techniques [73, 76]. Nevertheless, all proposed mechanisms fail to guarantee acceptable levels of privacy and accuracy at the same time [38].

To allow configurable per-query accuracy while providing a guaranteed upper bound on privacy loss, Dwork et al. introduced *Differential Privacy* (DP) [40]. DP is not a single mechanism but a theoretical framework to reason about the privacy provided by arbitrary systems. Privacy loss is thereby defined as the observable amount any output changes when a specific individual's data is modified or removed from the system. Noise addition to system outputs controls both privacy loss and analytic accuracy. Both are represented by the parameter $\varepsilon$ that determines the amount of noise to be added.

Naturally, choosing a specific $\varepsilon$ leads to a trade-off between privacy loss and analytic accuracy. Adding high levels of noise to any system output provides low privacy loss but impairs analytic accuracy and vice versa. At the same time the sum over all queries' $\varepsilon$'s, a cumulative $\varepsilon$, determines the total privacy loss any individual might have suffered. DP systems allow to set a limit, a privacy budget $\varepsilon$, count all queries against that budget and thus provide a guaranteed upper bound.

Unfortunately, DP systems tend to require large amounts of noise to guarantee chosen bounds on privacy loss and often miss to satisfy analytic needs [7]. Failure to make systems usable for analysts seems to be the reason why DP systems have not found wide-spread adoption in industry. Despite the availability of *formal* mechanisms with proven privacy guarantees, industry continues to use *ad-hoc* techniques of previous times. For example, online social media platforms employ simple privacy protections when they allow advertisers to obtain statistical results for a wide range of attributes over their users. Despite evidence of possible attacks [70], these interfaces cannot simply be removed, as they are at the core of major business models. Data analytics thereby is and will be a growing market for the foreseeable future. Businesses inside this market will only adopt as much privacy protection as is usable in accordance with their analytic requirements. Therefore, my research focuses on building data analytics systems that provide increased *usability* to analysts.

The goal of my research is to provide *usable* private data analytics systems that can be deployed in real-world analytic scenarios. To this end I follow two different approaches to improve on two usability challenges of DP. These challenges are (1) weakening DP's assumptions and guarantees to lower the noise while maintaining acceptable levels of privacy, and (2) selectively analyzing and using privacy budgets on parts of a dataset while keeping accuracy controllable and saving the budget on other parts. My approaches to these challenges are (1) doing completely away with DP guarantees and exploring the resulting possibilities for private analytics systems, and (2) using previously gained insights to improve DP and propose UniTraX, a solution to the second challenge.

**Usability over privacy.** In the first approach I start with the most usable yet privacy protection enabled system. Usability is ensured by requirements of unlimited queries and a publicly known low bound on noise added to answers. Privacy is not only protected by the noise added to answers but also by a defense-in-depth design consisting of different *ad-hoc* prevention and detection mechanisms that together counter different attack scenarios. Given the absence of *formal* methods there are no scientific guarantees, yet a security analysis over all practical attacks known to me shows them to be detectable if not preventable for an adversary without additional knowledge. Unfortunately, in the real world adversaries often do possess additional knowledge and attacks would succeed.

**Weak DP.** Building on my analysis of attacks I propose a hybrid system design that provides a weak version of DP. Weak in this context means probabilistic. The system provides DP but is allowed to fail in few cases. At the cost of perfect privacy protection the system is able to reduce required noise by a factor of 3.6. In the context of this system design I introduce *Query Based Bookkeeping* (QBB), a technique to keep track of privacy loss based on queries and their potential combinations. The novelty of the technique is its independence of the protected dataset. Using a real-world dataset of taxi rides in *New York City* (NYC) I show that the probability to be the victim of an unprevented attack is low. The insights on attacks and defenses gained with these system designs provide the grounds for Diffix [55], a commercially deployed analytics system that provides ad-hoc privacy protection.

**UniTraX.** Further building on the insights gained in the first approach, my second approach investigates potential improvements of DP mechanisms. In this context I present UniTraX [86], the DP successor of QBB. UniTraX allows analysts to query and analyze only a part of a dataset without limiting future analyses of other parts. In contrast to previous work [50], UniTraX is able to make introduced dataset bias observable to analysts and thus preserves statistical result accuracy. Biases are introduced when an analyst queries an important part of the dataset, depletes its privacy budget, and thus prohibits its use in any further queries. When other analysts do not know about this previous analysis and query the whole dataset their interpretation of answers is likely wrong, as they would not know that the important part of the dataset was not used to prepare their answer. UniTraX can block potentially biased queries and inform analysts about heavily queried parts of the dataset (see chapter 4 for details).

**UniTraX's usability.** UniTraX is designed with *formal privacy guarantees* in mind and there is a formal prove that UniTraX provides DP. To show its usability and general applicability I conduct multiple case studies on real-world datasets from different domains. In particular, I use one dataset each from the mobility, financial and medical domain, transform the data into the numeric format required by UniTraX, and perform different analytic tasks using UniTraX's privacy protecting analytic interface. I show that UniTraX saves significant amounts of privacy budget at moderate runtime overheads compared to previous differentially private systems. Saved budget is available for additional queries, which means that UniTraX allows more queries than previous systems while providing the same privacy protection and analytic accuracy.

**Contributions.** The contributions of this thesis are nine-fold.

1. A defense-in-depth system design (A) of ad-hoc mechanisms that provides privacy protection with guaranteed usability.

2. An attack analysis showing that (A) is able to defend against attackers that lack additional knowledge.

3. A system design (B) employing QBB and query constraints to achieve higher usability than previous work.

4. Mechanisms that allow (B) to provide more counting queries at the cost of weaker privacy protection.

5. A partial empirical validation on a real dataset showing that (B) is difficult to attack, despite weaker protection.

6. UniTraX, a system model and design that allows more queries than previous work, while making introduced biases observable.

7. A theoretical framework and proof that UniTraX provides DP.

8. An implementation of UniTraX that works across different domains.

9. An evaluation of UniTraX showing its ability to save significant amounts of privacy budgets at moderate overheads.

**Structure.** Chapter 2 presents system design (A) and its attack analysis. System design (B), its weaker protections, and its partial empirical validation are detailed in chapter 3. UniTraX is introduced in chapter 4 and evaluated in chapter 5. Finally, chapter 6 concludes and presents future work.

# Part I.

# Leaving Differential Privacy

# 2. Giving up on privacy guarantees

This chapter starts the first approach of my research. At its core, system usability is put before individuals' privacy protection. Giving up on privacy to preserve usability stands in contrast to most previous work, DP techniques in particular. However, reading critiques of DP techniques [7] it becomes clear that requirements of high noise and limits on the amounts of queries are two of the most important reasons for the lack of their adoption. To create a system that is easy to adopt I invert these requirements at the cost of privacy guarantees. Thus, my initial requirements are (1) low bounded noise and (2) unlimited queries.

Given the two initial requirements, DP techniques cannot be used to design a system as they require the opposite. Their requirements stem from the assumption of a powerful adversary with full knowledge of all individuals' data except for the individual of interest. In combination with low bounded noise and unlimited queries such adversary is always able to break the privacy of said individual. Simply repeating a query removes the noise added to answers after a finite amount of time. The knowledge of all other data allows the adversary to construct specific queries and corresponding answers to circumvent additional protective measures.

The requirement of bounded noise also precludes the use of any data modifying measures. Such techniques, e.g., $k$-anonymity, introduce noise into the original data through their privacy enabling data modifications. The system cannot tell the analyst the amount by which individuals' data was modified without breaking individuals' privacy. Statistical results over such modified data may then contain noise above any specific low bound.

Taking aforementioned conditions into account this chapter explores today's possibilities of ad-hoc techniques considering the various attacks that have been found. To prevent these attacks the presented system relies on a defense-in-depth approach combining multiple ad-hoc mechanisms. These mechanisms restrict the queries that the system may answer and monitor queries to suspend analysts when an attack is detected.

To evaluate the system's protective capabilities, the second half of this chapter investigates different kinds of attacks for different types of adversarial analysts. Unfortunately, the evaluation shows that in the general case only an adversary without any knowledge can successfully be defended against. In most cases there exist attacks which already work with few additional knowledge. In the general case the presented system is thus not able to provide acceptable privacy protection.

The chapter is structured as follows. First, an overview of the system design and its assumptions are presented in section 2.1. Second, all protecting and detecting mechanisms are listed in section 2.2, each with a short explanation of its purpose. Third, attacks for a variety of different assumptions of adversarial capabilities are evaluated in section 2.3 before I conclude in section 2.5.

## 2.1. Design overview and assumptions

The system makes a set of assumptions common to many private analytics systems. These are:

1. Individuals' data resides in a database with a single table.
2. Each row of the single table contains all data of a single individual.
3. All data is numerical.
4. Analysts (both benign and adversarial) can only access the data through a restricted and monitored analytic interface.

According to these assumptions the system resembles a database wrapper with control over any query going in and any answer coming out of the database. It is thus able to restrict queries and anonymize answers. Additional assumptions are:

5. Queries are restricted to counting, e.g., "SELECT count(*) WHERE x".
6. Conditions "x" may neither contain negations ($\neg$) nor disjunctions ($\vee$), only conjunctions ($\wedge$) are allowed.

Negations and disjunctions give an adversary with little knowledge about some individuals' data the flexibility to add known data into many query answers. The more known data answers contain the less anonymizing measures work. Thus, both these capabilities are forbidden. Their use can be simulated with multiple queries using only conjunctions at the cost of additional noise on answers.

## 2.2. Protection and detection mechanisms

The system uses a layered defense-in-depth approach. The following mechanisms are all applied together in order to prevent attacks. Actual counts $c_a$ from the database pass through all mechanisms before the last mechanism releases the final count $c$ to the analyst.

**Low count filter.** Uses thresholds $t_1$ (usually 1) and $t_2$ (usually 3). If the actual count $c_a$ is below or equal to $t_1$, i.e., $c_a \leq t_1$, it is suppressed and the returned count $c_r$ is set to 0. If $c_a$ is above $t_1$, i.e., $c_a > t_1$, a noise value is drawn from a Normal distribution with low standard deviation $\sigma$ (usually 1) and added to $c_a$, i.e., $c'_a = c_a + \mathcal{N}(0, \sigma)$. If the answer $c'_a$ is below $t_2$ it is suppressed and $c_r$ is again set to 0. Otherwise, $c_r = c_a$ is returned.

The low count filter prevents any query to count only a single or a small group of individuals. It prevents an adversary without any additional knowledge from attacking a single individual. All queries must count at least $max(t_1, t_2)$ individuals. Thus, all answers contain additional individuals' data. An attacker must know these individuals' data in order to be successful.

Through careful configuration of $t_1$ and $t_2$ the system administrator is able to control the minimum requirements for an attack.

**High touch suppression.** Skips exposed individuals while counting. Individuals are exposed if they have been counted significantly more times than others. The system stores the number of times an individual was counted. When a set of individuals is counted, the system determines the mean and variance of the previous counts of all individuals in the set with respect to a Normal distribution. If the probability for the number of counts is below a threshold for any individual, the individual is skipped and not counted.

High touch suppression prevents attacks that leverage additional knowledge about specific parts of the data. There, an adversary would separately query for each known part in combination with the same unknown individual. This attacked individual's data is then the only unknown component of each answer. By combining many queries and answers the attacker would finally obtain the individual's data.

Benign analysts are unlikely to use queries that count one specific individual unlikely many more times than others. Therefore, benign analysts do generally not experience additional noise in their answers. Analysts that trigger high touch suppression more than a defined threshold of times are temporarily suspended until further investigation by a system administrator. To allow for such investigation the system must store all queries.

**Fixed noise.** Noise added to output $c_r$ after low count filter and high touch suppression have completed. The noise depends on the records counted in the actual count $c_a$. The system hashes secret record identifiers of all records counted in $c_a$ and uses the hash $h$ as seed to a Normal distribution. The random value $v$ drawn from this distribution is added and $c'_r = c_r + v$ is forwarded to the next layer. The standard deviation of the Normal distribution is chosen low (e.g., $\sim$1.41).

Basing the random noise on the set of counted records prevents attacks that repeatedly count the same records. In such an attack the adversary would try to average away any added noise. Queries would be syntactically different but semantically identical. Therefore, these queries would all count the same set of individuals. In case of independent noise, the noise values would be different for all answers and gradually average away. With fixed noise, however, the same set of individuals always leads to the same noise for an answer. The noise can thus not be averaged away by repeatedly counting the same set of records.

Benign analysts should not have to count the same set of records, i.e., individuals, twice. The system therefore counts the occurrences of each set of records for each analyst. Once a threshold is reached for any particular set, the respective analyst is temporarily suspended until further investigation by a system administrator.

**Random noise.** Noise added to resulting count $c'_r$ of the previous layer. The noise is drawn from a Normal distribution initialized with a random seed. Another random value $v'$ is added to the output of the previous layer and the answer $c''_r = c'_r + v'$ is handed to the next layer. Again, the standard deviation of this noise is chosen low. If both fixed and random noise choose the standard deviation $\sim$1.41, the combined standard deviation is $\sim$2.

Using random noise in addition to fixed noise prevents an adversary from directly recognizing whether two counts stem from the same set of individuals. With only fixed noise the released answer would not change if the same set of individuals was counted twice. For different sets, however, the answer would change. With random noise answers always change independent of the counted set of individuals.

**Rounding.** Takes the count $c''_r$ of the last layer, rounds it to the nearest step $s$ (usually 5) and returns the final count $c$ to the analyst.

Although rounding is just a different kind of noise, it requires additional queries to remove, while having only a limited effect on output accuracy. An adversary needs additional knowledge to interpret the meaning of final counts being rounded to different values.

## 2.3. Attacks

In an iterative process the just presented system design with its different layers of defense and detection mechanisms is improved with the help of different attacks. Increasingly severe assumptions about the adversaries' powers are used as follows.

### 2.3.1. Adversaries

At the very minimum any adversary needs to know some condition that is unique to the individual they want to attack. Without such knowledge there is no way an adversary can succeed. Without knowing anything about an individual it is not possible to know whether the individual will be counted by a specific query.

Next to this basic attack knowledge adversaries might know about the data of different numbers of other individuals. In general, the more they know the better they can attack. The different adversaries to consider are thus:

- *No additional knowledge.* The adversary knows enough to uniquely identify and count the victim but no data of other individuals.
- *Few additional knowledge.* The adversary also knows some of the other individuals' data.
- *Substantial additional knowledge.* The adversary knows the data of many but not all other individuals.
- *Worst-case additional knowledge.* The adversary knows all other individuals' data.

Beyond knowledge of other individuals' data an adversary might also be able to create, modify, or delete data. However, the following paragraphs present generic attacks that are possible for the two weakest adversaries without such additional capabilities. Before the system can successfully defend against all attacks of a weak adversary, it is not necessary to investigate more powerful adversaries.

| | Comparison set | Attack set | | |
|---|---|---|---|---|
| Possible size | $n$ | $n-1$ | $n$ | $n+1$ |
| Combination 1 | $I$ | | $I$ | $I \cup \{v\}$ |
| Combination 2 | $I \cup \{v\}$ | $I$ | $I \cup \{v\}$ | |

Figure 2.1.: Sets used in attacks

### 2.3.2. Strategies

To obtain information about a single individual, e.g., whether the individual's data is in the database, an adversarial analyst must use queries that are able to circumvent all defense and detection mechanisms. As the interface is limited to counting individuals whose data matches specific conditions, the adversary needs to be able to infer from counts whether the targeted individual got counted.

The general attack strategy for inferring information through counts is to create two counts, (1) a controlled comparison count and (2) an attack count. For the first count the adversary knows whether the victim would be present or absent. The second count then challenges this knowledge. Depending on the outcome of the second query it becomes clear whether the victim is actually present. It is not necessary to know the true counts for the attack to work. The adversary only needs to be able to determine whether the counts differ.

The simplest way would be to only count $v$, which does not work because of the low count filter. Instead, each count needs to include additional individuals. These must be part of a controlled set $I$ for the attack to work as explained. Figure 2.1 shows the different possible combinations of comparison and attack set. For the comparison set an adversary knows whether it includes $v$. For the attack set it is to be found out. More formally this works as follows.

**Combination 1.** The adversary is able to devise a query that counts $I$, i.e., a query that does not count $v$. This means that the answer to the query is so constrained that $v$ is not counted. To learn something about the victim it must be allowed to appear and one needs to be able to measure whether it did. To this end, one needs to create an attack query that only allows the victim $v$ to additionally match. The easiest way to create such an attack query is to modify the condition $c$ of the comparison query, remove a part, and obtain condition $c'$. The adversary needs to be sure that $c'$ does not allow any other new individual to show up except potentially $v$. Through comparison of the answers it is then obvious whether $v$ actually showed up and thus participates in the database.

**Combination 2.** In this combination the adversary knows that $v$ is present in the dataset and is able to generate a query that counts $I \cup \{v\}$. In order to obtain new information about $v$, an attack query must then result in either the same answer or one where $v$ is removed. Similar to before one can simply modify the initial condition $c$ but this time add a new part to create $c'$. Of $c'$ the adversary must know that it does not constrain $I$, i.e., $c'$ matches all individuals in $I$. The answer to the modified query will either be $I \cup \{v\}$ or just $I$. Comparing with the initial answer allows the adversary to find out whether $v$'s data matches the new condition $c'$.

Because our fixed noise is the same for equivalent sets, one can simply compare the answers once the random noise and rounding has averaged away. In order to detect an adversary who averages away the added random noise and rounding, we track the number of occurrences per unique set of individuals. Once a threshold has been reached we suspend analysts for further inspection. However, if adversaries know the exact thresholds of the detection mechanism, it is possible for them to stop their attack before it can be detected.

## 2.4. Attack implementation

The attack strategy presented in the previous section explains why the system does not allow negation and disjunction in query conditions. With those capabilities even an adversary of the weakest assumptions, i.e., with no knowledge of other individuals' data, could easily devise queries for comparison and attack sets. Banning the use of $\neg$ and $\vee$ operators thus increases the difficulty to create suitable attack queries. Unfortunately, negation and disjunction can often be simulated through semantically identical queries without these operators. The next paragraphs provide a more detailed look at the implementation details and explain how attacks on the system might be implemented.

**Conditions.** An analyst provides a condition $c$ in a query to select the set of individuals counted by the query answer. Query execution matches $c$ to all individuals' data in the database and returns the number of matching individuals. To match a condition the attributes of an individual's data must match the analyst provided values in the condition. A single attribute gets matched to a single value by binary operators $==, <, >, \leq, \geq$.

**Isolating condition.** In order to attack an individual an adversary needs to be able to uniquely identify that individual's data with a query condition. For example, the adversary might know that the victim individual is the only employee in a company's database with a salary of \$300,000.00. An isolating condition would then be "salary $==$ 300000".

**Attacking with combination 1.** By using an isolating condition an adversary is able to create a query that counts some individuals $I$ other than the victim $v$. The simplest way—negating the condition—is prevented by the ban of the $\neg$ operator. However, it is still a simple task to come up with a suitable query condition, e.g., "salary $<$ 300000". This query counts all employees with lower salaries but not the victim with isolating condition "salary $==$ 300000". The attack query condition is then "salary $\leq$ 300000".

**Cheating same set detection.** If it were possible the adversary would just repeat the two queries until the random noise and rounding has averaged away and the different answers could be compared. However, the same sets of individuals would be counted multiple times and depending on the threshold setting that point might not be reached. Nevertheless, there are different strategies the adversary can use in order to avoid detection.

An adversary without knowledge of other individuals' data can perform the attack counting different sets by using a moving split. In that scenario the attacker uses an additional part in query conditions that splits each query into two half queries. For example, the adversary might choose a condition on month of birth to create 12 different splits. Condition "salary $<$ 300000" then becomes 24 separate conditions "salary $<$ 300000 $\wedge$ month-of-birth $\leq$ 1", "salary $<$ 300000 $\wedge$ month-of-birth $>$ 1", etc., each of which can be repeated threshold-1 times without detection (assuming a large enough dataset where multiple employees have birthdays each month).

**Attacking with combination 2.** Here, the adversary knows that victim $v$ is part of the database. The goal is to learn additional data about the victim. The comparison query for combination 2 must include the victim so the adversary might simply reuse condition "salary $\leq$ 300000". However, in order to create an attack query the adversary requires additional knowledge about the other employees counted in that query. This knowledge is necessary to devise a condition that cannot exclude anyone except potentially the victim. For example, if the adversary is certain that no other employee with a lower salary owns a helicopter the respective condition to test whether the victim does is "salary $\leq$ 300000 $\wedge$ helicopter $==$ 0". If the second answer differs from the first $v$ has a helicopter. Again, one can only compare the answers after the removal of random noise and rounding, which works as before.

**Cheating high touch detection.** In both above presented attacks all counted individuals are counted roughly the same number of times. Never is there an individual that is counted significantly more times than other individuals. To ensure that all individuals in the database are counted exactly the same number of times one can also add additional queries that count all remaining individuals. For example, if there is a query with condition "salary $<$ 300000" one just adds another query with condition "salary $\geq$ 300000". That way all individuals are always counted exactly the same number of times and high touch suppression and detection fail to prevent and detect the attack.

**Variations on attacks.** Next to directly learning some new fact about an individual there are similar attacks with different goals. The simplest is to just learn a true count. The adversary is interested in learning the count of a set of individuals, e.g., how many people have a higher salary. Another variation is learning value steps. There, the adversary wants to know, for example, the different salaries the company pays its employees and what the distances between them are. The attack would continuously increase the salary range in its query conditions. Whenever a new fixed noise is detected, a new salary step is discovered. Example conditions would be "salary $\leq$ 10000", "salary $\leq$ 10001", "salary $\leq$ 10002", etc., however, same set detection might not be as easily cheated with such small value steps.

## 2.5. Conclusion

In this chapter we explored the possibilities of a useful private data analytics system. Useful in this context meant unlimited queries with strictly low bounded noise on any returned statistical results. Despite severe restrictions on the query interface and multiple layers of ad-hoc defense and detection mechanisms even the weakest attackers are still able to launch successful attacks. Although there might be datasets for which these protections would suffice our system design does not work as the sought general solution for private data analytics. Nevertheless, we take away the following insights.

1. High touch based mechanisms can easily be cheated.
2. Attacks work with analyst chosen value ranges in conditions.
3. Different sets of individuals are simple to create by splitting the same set many times along different dimensions (month-of-birth in our example).
4. Depending on the system's choice of noise in combination with rounding, a single query can be more or less effective in an attack.
5. Attacks on additional information are hard, as specific sets of individuals—matching the sought information—must be known to the adversary.
6. The sets known to the adversary must be "next" to the victim in the dataset so one can include or exclude the victim in comparison and attack queries.

From these insights we derive multiple paths of future work.

- Investigate restrictions on value ranges in query conditions (Insight 2).
- Investigate mechanisms to prevent the use of splits in attacks (Insight 3).
- Investigate whether it helps to track the actual protection the system provided in a specific execution of a query (Insight 4).
- Investigate tracking queries and preventing attacks by stopping once a threshold on "bad" queries has been reached (Insight 4).
- Investigate detection of queries that are "next" to each other (Insight 6) and its effect on adversaries with limited knowledge (Insight 5).

While I follow the last two paths in the following chapter of this thesis, the first two paths provided the grounds for development of the Diffix system [55].

# 3. Towards low noise and large budgets in privacy preserving analytics

In chapter 2 I gave up on privacy guarantees in favor of usability for analysts. I defined usability to be low bounded noise per query and unlimited queries. However, the system design I came up with was not able to fully prevent attacks for adversaries with limited knowledge of other data in the protected database. I believe that in most real scenarios certain knowledge about the data in the protected database is available to adversaries. This might simply be the case because common knowledge applies. However, I also believe that in reality the all-knowing adversary is unlikely to exist and thus most if not all attacks are performed based on partial knowledge.

Continuing with what I started in chapter 2 I still try to answer the questions: Can one draw on both *ad-hoc* and *formal* research threads to build a system that is on one hand relatively general and easy to use, while on the other hand satisfies analytic needs? To the extent that one can, what trade-offs are necessary, for instance, in terms of weakened assumptions or system constraints? As a second step in answering these questions, the following chapter presents a system design, informal analysis of that design, and a partial validation of the design based on a real dataset (the NYC taxi dataset). This chapter was written in collaboration with my adviser Paul Francis and thus continues in we perspective.

The starting point in our design is the *Provenance for Personalised Differential Privacy* (ProPer) [50] system, a DP system that increases the effective privacy budget by applying per-user budgets. In ProPer, a user's budget is only reduced if the user's record contributes to a query's answer, or put another way, if the user's record passes the filter conditions of the query. This increases the effective overall budget because queries with disjoint filter conditions deplete different users' budgets. When a user's budget is depleted, the user's record is silently

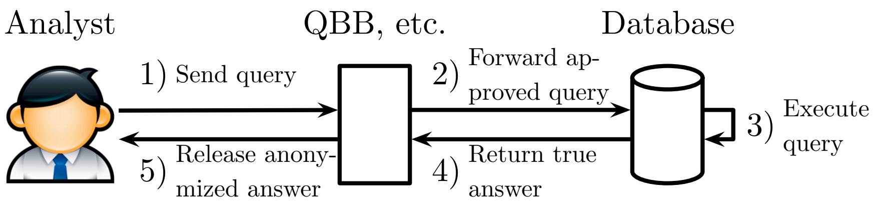


Figure 3.1.: High-level system design of our system, which follows the concept of a database wrapper. The example shows a successful query and its respective answer passing through the system. An analyst sends the query to our system (1). After approval the query is forwarded (2) to, and executed by the database (3). The true answer is returned to our system (4), which releases the anonymized answer to the analyst (5).

dropped from the database. However, dropped records might distort answers without the analyst being aware of the distortion.

To alleviate this problem, while still maintaining the benefits of disjoint budget tracking, our design uses *query-based budget tracking* (called Query Based Bookkeeping, QBB) rather than user-record based tracking. By way of example, suppose a query counts the number of users between ages 10 and 20. ProPer would decrement the individual budgets of each user that falls in that age range. By contrast, QBB keeps track of the budget for the the age range itself. As a result, the analyst knows how budget is reduced for each query. Instead of giving distorted answers as with ProPer, QBB simply blocks queries where part of the range has a depleted budget. This comes at a cost: tables are limited to numerical data, and the system must maintain per-query state.

Although QBB and ProPer behave differently, they provide the same budget overall. Separate from QBB, our system introduces new mechanisms that further increase the budget by a factor of 3.6 by exploiting the assumption that attackers have only partial knowledge of the database—an assumption that is realistic in many scenarios. These mechanisms include a constraint on queries that limits analysts to one range per column per query, checks that prevent analysts from repeatedly generating identical answers, and the use of Gaussian rather than Laplacian noise. This comes at the cost of opening the system to possible but “hard-to-launch” attacks.

All our proposed mechanisms are part of a system design that follows the high-level concept of a database wrapper. Such wrappers intercept any queries going into and any answers coming out of the database holding the users' data. Figure 3.1 shows this high-level concept, where our system with QBB and the other mechanisms sits in between the analyst and the database. Any query going into the database is thus approved first, while any answer released to the analyst is anonymized. At both points the system can intervene in order to protect users' privacy.

To better understand the amount to which users' privacy in our system is susceptible to any of the remaining types of attacks we undertake a case study based on the NYC taxi dataset. This study shows that exploiting our system design with direct or observed knowledge about users' data is difficult. We verify these findings across adversaries of different strengths all the way to a realistic worst-case adversary. Using the taxi data we demonstrate for each assumed attacker the amount of knowledge that is necessary for a successful attack and thereby empirically verify that attacks are indeed hard to perform in practice.

Our contributions in this chapter are threefold:

- A system design that improves the usability of ProPer at the expense of constraints on queries' filter conditions.
- Mechanisms that increase the budget by 3.6x for counting queries at the expense of opening the system to hard-to-launch attacks.
- A partial empirical validation on a real dataset showing that attacks are indeed hard to launch in practice.

We first illustrate the utility problems with ProPer in section 3.1, before we introduce QBB in section 3.2. We describe the mechanisms that allow to increase the budget for counting queries in section 3.3 and present the complete system design in section 3.4. The empirical evaluation and case study we show in section 3.5, before we conclude with future work in section 3.7.

## 3.1. Utility of ProPer

ProPer allows significantly more queries per budget than previous DP based systems, which typically manage the budget at a global level. Global budgets must be diminished by every query answered by the system independent of which users' data, i.e., records, are read by the query. By keeping a separate budget per record, ProPer is able to run queries that have disjoint filter conditions in parallel without diminishing any single budget twice. Thus, partitioning the data with disjoint filter conditions into $n$ parts allows $n$ times as many queries as before.

Dynamic streaming databases are the use case for which Ebadi et al. claim to provide acceptable utility, however they note that it remains unclear how much utility ProPer is able to provide in case of a static database. For the static case we find that the focus on personal budgets and the generality of allowed transform and filter operations impair utility of the system. In the following paragraphs we first illustrate these problems with two examples. We then propose a system design based on uniform budgets and QBB that provides utility without compromising privacy.

To protect each users' preferred level of privacy, ProPer tracks each individual user's chosen budget for the user's record, diminishes the budget when a query reads the record, and silently drops the record when its budget is depleted. However, due to the randomness of user chosen privacy levels at any point in time the system potentially drops an unknown number of records and thus changes the base number of records that is read by queries. Results of queries therefore turn into arbitrary numbers and are useless. For instance, assume the following example.

**Example.** An analyst explores the salary structure of a company and wants to obtain scatter plots of salary ranges versus other dimensions, e.g., amount of holidays taken. She creates filter conditions for each combination of range on salary and range on the respective other dimension, e.g., salaries from \$40k to \$50k and 5 to 10 holidays taken. With each such filter she counts matching employees and successively obtains the data for all her plots. Although the sum of all counts per plot should be within noise bounds, they decrease with each successive plot.

Total counts successively decrease as over time records have been dropped silently from the database, which cannot be revealed to the analyst. Therefore it is now unclear which counts in the analysis are based on how large a part of the originally complete set of employees. Some budgets might have been too small for the cost of even a single count, others too small for the cost of two counts, and so forth. However, the analyst has no way to determine which employees contribute to each count and therefore the counts received from the system are of low utility to her. In summary, many statistical outputs of the system have low utility when users' preferred levels of privacy are configured.

To increase utility and prevent problems with personalized budgets we use initially uniform budgets across all records, the amount of which can be publicly known. Knowing the initially available budget allows the analyst to plan ahead and adjust her behavior. In particular, she might ask for fewer salary groups with more employees per group, which allows for higher noise on the counting queries and thus a lower budget cost per query. By adjusting queries to known budgets analysts thus achieve high utility within the given limits on noise and budget.

Unfortunately, achieving high utility not only depends on knowing the initial budget given to records but also on keeping track of any remaining budgets. At this point tracking budgets is a cumbersome manual process, as ProPer cannot tell the remaining budgets after using the expressive filter and transformation functions it allows. Any union-preserving function can be applied to transform or filter the data, however it takes additional effort to determine if and how such function partitions the data. For instance, assume the following example.

**Example.** The analyst uses automated tools to find clusters in the salary data, to create custom filter functions for each cluster, and to perform additional analyses for each cluster. She knows the uniform initial budgets and is sure that even if clusters overlap budgets will not be depleted. Nevertheless, to determine the exact remaining budgets for successive analyses she needs to manually inspect the sequence of queries made by used tools and also determine how the custom filter functions partitioned the data. She must find out which sets of queries got asked about each partition of the data and how much budget therefore remains for each.

In summary, it is difficult for analysts to keep track of remaining budgets despite the use of uniform initial budgets. However, without careful bookkeeping there is the risk of silently dropped records that reduce utility of statistical results. We conclude that there should never be the case where records are silently dropped from the database. In contrast, the analyst should be able to plan her analysis around how much budget she uses and how much budget remains given her previous queries.

## 3.2. Query Based Bookkeeping

To mitigate analysts problems with tracking budgets we propose *Query Based Bookkeeping* (QBB), a mechanism that tracks budgets exclusively based on queries and not on records. It allows analysts to exploit queries with disjoint filter conditions, but unlike ProPer it further allows analysts to stay informed about remaining budgets and thus better plan their future queries. All queries are known to analysts and thus only public information is used in this mechanism to make any decisions about how much budget remains. Therefore, when analysts pose new queries QBB can decide to reject select queries where remaining budgets are too small, but allow others that still fit. None of these decisions are based on the records in the database and thus no private information is leaked by knowing about what and how the mechanism decided.

In this section we explain how QBB protects users' privacy, how we simplify its design, how the mechanism actually tracks budgets, and how the budget cost of a query depends on its parameters. Instead of tracking the total budget costs for sets of queries actually reading the same records as ProPer does, QBB tracks the total budget costs for sets of queries that *have the potential* to read the same records. We thus create a strictly more restrictive mechanism in order to be able to base it exclusively on queries and their respective parameters.

The most important query parameter for budget bookkeeping is the filter condition, which determines whether a user's record is read by a query or not. We know that queries with disjoint filter conditions cannot read the same record, and inversely queries with non-disjoint, i.e., intersecting, filter conditions potentially read the same record. In fact, any record matching such intersection of filter conditions is read by all respective queries and potentially suffers their combined privacy loss. Therefore, QBB tracks budgets for all possible sets of queries with intersecting filter conditions, which for any record in the database includes all sets reading that record. Rejecting new queries if they deplete the budget of any single such set thus protects all users' privacy without using any information about the actual records in the database.

To be able to determine when filter conditions intersect we restrict their expressiveness to ranges over numerical data. We further simplify our design by only allowing a single range condition per data field of a record. Joining a set of such ranges across multiple columns results in a single bounding box that determines which records pass and can be read by a query and which cannot. Non-disjoint ranges or bounding boxes then intersect in some areas and records in those areas are read by all respective queries. Figure 3.2 shows examples of such filter conditions for one- and two-dimensional data and for both scenarios the figure indicates the area of intersection, where records are read by all queries. The same principles work for higher dimensional records.

In both examples of Figure 3.2 QBB tracks budgets for the four subsets $\{\{\}, \{q_1\}, \{q_2\}, \{q_1, q_2\}\}$ of queries. It is obvious how any record must be covered

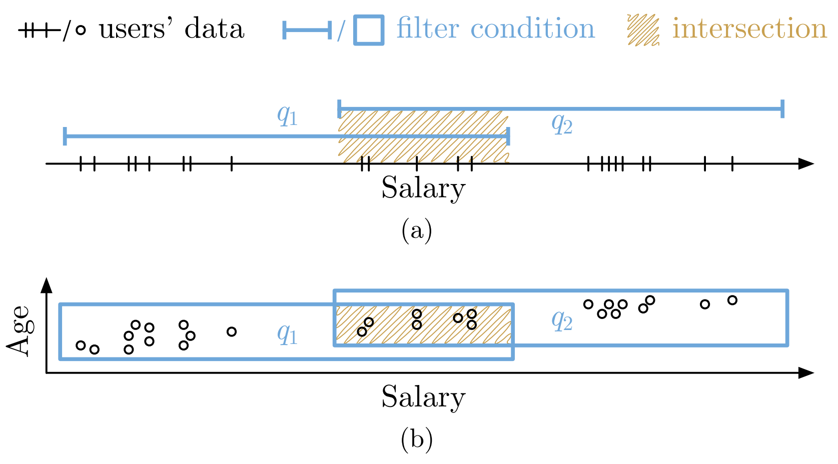


Figure 3.2.: Examples of filter conditions of two different queries $q_1$ and $q_2$ for one-dimensional (a) and two-dimensional (b) user data. In both cases shown filter conditions are not disjoint and therefore intersect. We only allow filter conditions that consist of a single range per data dimension, which results in a single bounding box filter condition per query.

by one of these four. Therefore, tracking budgets for each subset is equivalent to tracking budgets for any record with the advantage that tracked budgets are public and can be used by analysts to plan ahead and obtain statistical output with high utility.

However, as already mentioned, the advantages of QBB come at the cost of restrictions to expressiveness. These restrictions are enforced by the query interface of our system, which only accepts queries following the rules shown in Figure 3.3. The rules ensure, for example, that no more than a single range can be defined per column and thus queries follow the rule of a single bounding box. If a query does not define a range for a column, the system automatically assumes a wildcard range, i.e., $[-\infty, \infty]$. Columns, transformations, and query types thereby need to be pre-approved and installed by an administrator, who ensures that all transformations have a single numeric dimension as output and all query types have correct parameters specified for their results to be properly anonymized. The administrator further adds the output dimensions of transformation functions to the set of allowed dimensions, e.g., the transformation that calculates day-of-week of the entry date of an employee results in the day-of-week-of-entry dimension to be allowed in the definition of a filter range. Similarly, new query

$$
\begin{array}{lll}
\langle \textit{lowval} \rangle & ::= & \langle \textit{decimal} \rangle \mid `\texttt{min}'; \\[1ex]
\langle \textit{highval} \rangle & ::= & \langle \textit{decimal} \rangle \mid `\texttt{max}'; \\[1ex]
\langle \textit{range} \rangle & ::= & `\texttt{low}' \; \langle \textit{lowval} \rangle, \; `\texttt{high}' \; \langle \textit{highval} \rangle; \\[1ex]
\langle \textit{col-con}_c \rangle & ::= & `\texttt{column}\ c', \; \langle \textit{range} \rangle; \\
 & & \text{with } c \in \{1, 2, ..., n_c\} \\
 & & \text{and } n_c = \# \text{ of columns} \\[1ex]
\langle \textit{col-con-list}_i \rangle & ::= & `' \mid \langle \textit{col-con}_i \rangle, \; \langle \textit{col-con-list}_j \rangle; \\
 & & \text{with } i > j \\[1ex]
\langle \textit{tran-con}_t \rangle & ::= & `\texttt{transformation}\ t', \; \langle \textit{range} \rangle; \\
 & & \text{with } t \in \{1, 2, ..., n_t\} \\
 & & \text{and } n_t = \# \text{ of transformations} \\[1ex]
\langle \textit{tran-con-list}_k \rangle & ::= & `' \mid \langle \textit{tran-con}_k \rangle, \; \langle \textit{tran-con-list}_l \rangle; \\
 & & \text{with } k > l \\[1ex]
\langle \textit{condition} \rangle & ::= & `\texttt{filter}' \; \langle \textit{col-con-list}_r \rangle, \; \langle \textit{tran-con-list}_s \rangle; \\[1ex]
\langle \textit{query} \rangle & ::= & `\texttt{query}\ q', \; \langle \textit{condition} \rangle, \; `\texttt{noise}' \; \langle \textit{decimal} \rangle; \\
 & & \text{with } q \in \{1, 2, ..., n_q\} \\
 & & \text{and } n_q = \# \text{ of query types}
\end{array}
$$

Figure 3.3.: Rules for queries accepted by the query interface in BNF

types must be appended to the list of allowed types to be available for analysts to use.

Although we severely restrict the interface compared to ProPer, we believe these restrictions are necessary for useful analytics. In any event, more complex transformation and filter functions *can still be run as part of a query.* The analyst only needs to ask for the approval of a new query type that combines the complex filter function with the actual code of a previous query and thus creates a new combined type of query. Furthermore, analysts can combine the results of multiple queries with disjoint filter conditions in order to simulate the operations of logical OR or NOT, which both are effectively prevented by the restriction of a single bounding box. For instance, assume the following example.

**Example.** To analyze the salary structure of departments at the west coast the analyst can separately analyze each state and combine the results herself or she creates a new type of query that reads all records and internally filters them before computing its final output. The former only affects budgets on the west coast partition of the data but leads to multiple times noise added to the combined answer as the result of each state includes noise. The latter affects all budgets but only includes noise once in the final result. A third option would be to introduce a new transformation function that maps records to 0 or 1 depending on whether they are west coast or not and then use regular filter conditions on that new dimension.

Analysts have a choice between higher noise, wasted budget, and impaired performance. They can either use a combination of multiple queries with different filters each of which increases the noise on the final result. As a second option they might execute everything in a single combined query with a wildcard filter in which case budget is potentially wasted. Or they introduce a new dimension into the bookkeeping mechanism, which increases the dimensionality of tracked filter conditions. High dimensionality, however, might negatively affect future query execution times. The optimal choice of method depends on the structure of the data and the number of transformation functions and query types that are already present in the system.

It is the restrictions and simplifications on queries that enable QBB to understand which sets of queries read the same records. Nevertheless, in order to find all sets of previous queries intersecting with a new query, QBB needs to keep a history of all queries. In addition to space requirements, such history poses different challenges with respect to performance. One way to exploit the spatial

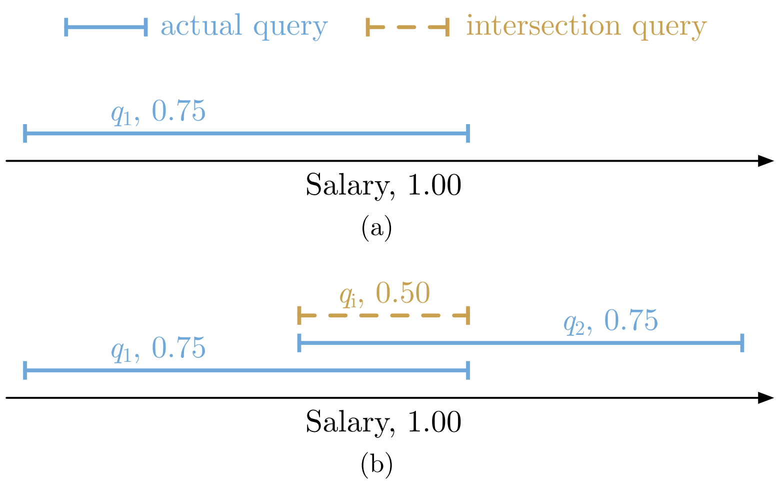


Figure 3.4.: Intuition of the history of QBB. We assume that the total allowed budget is 1.00 and that each query incurs a cost of 0.25. Figure (a) shows the state of the history after the first query is added into the system and its remaining budget is set to 0.75. Figure (b) shows the new state after the second query is added with its budget set to 0.75 respectively. The intersection of both queries has a lower remaining budget and requires an intersection query with a budget of 0.50. The terms query and filter condition are used interchangeably here.

nature of queries in implementing the system is to use a spatial index structure, e.g., a R*-tree or related structure [11]. To keep inspection and approval times for queries short one can use precomputed remaining budgets. One might further use a search technique that requires only a single scan of any relevant history entries to find the intersection of the new query with the lowest remaining budget.

The history structure works similar to the intuition provided in Figure 3.4. Here the terms query and filter condition are used interchangeably, so when we refer to a query most of the time we actually refer to its filter condition here. In the figure one can see how queries are stored preserving their spatial nature. The example shows the two different states of the history while two queries are added. Each query carries the highest remaining budget it can offer to any new query. In the example the cost of each query is assumed to be 0.25, while the overall total budget at any point is 1.00, which is indicated on the x-axis. For any intersection between queries where a query is equal to the intersecting area, i.e., where one query is completely subsumed by another query and thus covers the total intersecting area, no additional intersection query must be added in order to store the remaining budget for the intersection. In the example however that is not the case and an additional intersection query with a budget of 0.50 must be added to the history.

To determine the budget cost of a query we use the theory of DP, in which budget and cost both are represented by $\varepsilon$. The theory defines a fixed relationship between the worst case budget cost $\varepsilon$, the noise added to the answer, and the sensitivity, i.e., the worst case output change with respect to a single record change, of a query. We use this relationship to determine the worst case cost given the noise level requested by the analyst in the query and assuming that the database administrator correctly verified the claimed sensitivity of the requested query type. As the sensitivity is fixed, cost and noise directly correlate with each other. This allows analysts to pre-determine the cost of their queries, as they know the query types and the amounts of noise they are going to request.

In summary, this section answers the four main questions of how QBB protects privacy, how it is more restricted than previous systems, how it keeps track of budgets, and how the budget cost of a query is determined. It details why privacy protection is based on statefully tracking $\varepsilon$-costs for sets of queries with intersecting filter conditions that are restricted to single ranges over numerical data dimensions. At an informal level this section also shows how tracking budgets for all possible records is strictly more restrictive than ProPer, which only tracks budgets for existing records. We therefore propose to replace the record based protection mechanism by the strictly more restrictive QBB and remove un-necessary redundancy. QBB is thereby fully compliant with the theory of DP and can provide a guaranteed upper bound on privacy loss. At the cost of restrictions to the expressiveness of filter and transformation functions we arrive at a design that provides predictability and increased utility to analysts.

## 3.3. Increasing budgets for counting queries

QBB allows larger budgets when queries partition the data, which conforms to the theory of DP. In general it contradicts this proven theory to be able to increase budgets and keep the noise without suffering from a significantly increased risk of privacy loss. Nevertheless, in this section we exploit QBB to double the budget for counting queries at the cost of opening the system to hard-to-launch attacks. After giving the high level intuition we present an investigation of possible types of attacks, during which we introduce three additional protection mechanisms, *Query Similarity Detection* (QSD), *Cheater Detection* (CD), and *Gaussian noise*. Together, these prevent many and impair the remaining attacks. In the end, we discuss which powerful attacks prevent us from increasing utility further.

We consider an attack to be "hard-to-launch" when an attacker requires knowledge of some of the victim's data as well as knowledge of data points "around" the victim in order to achieve even a small probability of learning something about the victim with high confidence.

Raising utility despite the theory of DP requires weakening the assumptions of that theory. An interesting idea in that direction is Noiseless Database Privacy [8, 14], a theoretical concept that allows to reduce added noise, if the input data of a query already contains uncertainty, i.e., noise in the form of records unknown to analysts. For such records to exist the authors weaken the assumptions of DP and only assume attackers with partial knowledge. This is reasonable in many real world scenarios with static databases. We also believe that many queries have filter conditions that include records unknown to an analyst. However, as we show later in this section, certain types of attacks remain powerful and thus completely removing artificial noise is not an option.

Before we look at attacks we want to first talk about the effect of uncertainty on answers to queries. In general it is not clear how much noise such uncertainty adds to an answer. Assume a record with a severe outlying data value. A sum over the column containing this value is then dominated by the outlier, which is easy to spot and thus makes the noise of the unknown records insignificant. As a simplifying design decision we therefore constrain ourselves to counting queries,

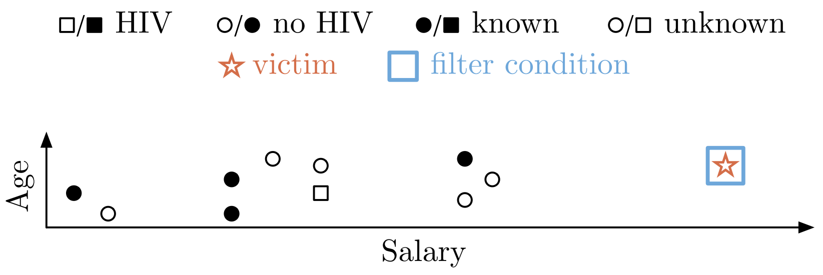


Figure 3.5.: Example of filter condition for directly asking for the data as well as repeating the same query multiple times. For simplicity the filter condition does not indicate its implicit "no HIV" setting.

where answers are the least dependent on any data values in records, as they simply count the number of records passing the filter condition.

For the following investigation of attacks we assume a system that employs QBB and allows twice the budget for counting queries than would be allowed according to DP. Additionally we assume that any single query is only allowed to consume up to a certain amount of budget, e.g., 25%, so multiple queries are needed to consume the total budget. For a successful attack it is then necessary to use multiple counting queries without uncertainty about the counted data, i.e., only known records plus the victim are counted. To defend against these attacks the system must ensure that unknown records are included in any such query. In the following paragraphs we present different types of attacks on the assumed system. We start with the simplest and end with the most sophisticated type we could formulate. In between we introduce *Query Similarity Detection* (QSD), *Cheater Detection* (CD), and *Gaussian noise* to successively upgrade the system to better defend against these attacks.

### 3.3.1. Query Similarity Detection

At this point we revisit and work from the previous example of the analyst investigating the salary structure of her company. However, this time we assume the following additional circumstances. First, employee records also contain the employee's HIV status. Second, the analyst knows the age and salary of some of her colleagues and also knows that there are no other employees with the same data. Third, for one of the otherwise known colleagues she does not know the HIV status and wants to find out.

**Attack.** The simplest attack the analyst can conduct is to simply ask for the data. She thus prepares a filter condition with ranges for the known age and salary and additionally a range that only matches the numerical representation of "no HIV". If the system returns a count of 1 the colleague in question is indeed HIV positive, otherwise the count is 0, which indicates the colleague is HIV negative. We can see a depiction of this attack in Figure 3.5. For simplicity the shown filter condition does not indicate the setting "no HIV".

Unfortunately for the analyst we have not completely removed artificial noise and additionally have a limit on the budget cost per query, which results in a lower bound on the noise the analyst might choose. Therefore, the answer to her query might be 0 or 1 but also $-1$, 2 or any other number with some probability. To overcome this noise the analyst must obtain the same count with different noise values multiple times so she can average away the noise. As the noise is drawn from a symmetrical distribution, averaging over enough samples reveals the mean, i.e., the true count.

**Attack.** To repeatedly obtain the same result with different noise, the analyst repeats the previous attack query. Filter conditions are the same for all queries.

The problem with this attack is the fact that the analyst is able to use the same query and filter condition throughout the attack. The effort of finding a suitable condition thus only needs to be rendered once. To counteract this attack opportunity we propose *Query Similarity Detection* (QSD), a stateful mechanism that detects and prevents the repeated use of the same or similar filter conditions. An attacker then has to find multiple different filter conditions that all fulfill the attack requirements and only allow known records and the victim to be counted. Such requirement significantly reduces the number of cases where an attack is possible and is one of several mechanisms that make attacks hard to launch.

Filter similarity is defined per data dimension of two queries. If the ranges in any dimension do not intersect, the queries are not similar. Otherwise, they are similar if for all dimensions the ratio of intersecting range $i_D$ to total range $t_D$ is above a certain threshold, e.g., 90%. Figure 3.6 shows two example scenarios of comparing a new to a previous filter condition. In Figure 3.6a the ratio $100 \cdot \frac{i_D}{t_D}$ is above 90% for both dimensions $D$ and the filter conditions are determined to be similar. In Figure 3.6b the ratio for the "Salary" dimension is below 90%, which results in the filter conditions not being similar.

Not only to impede attacks but also as a convenience to the analyst, we propose for QSD to return any found duplicate or similar query and its corresponding

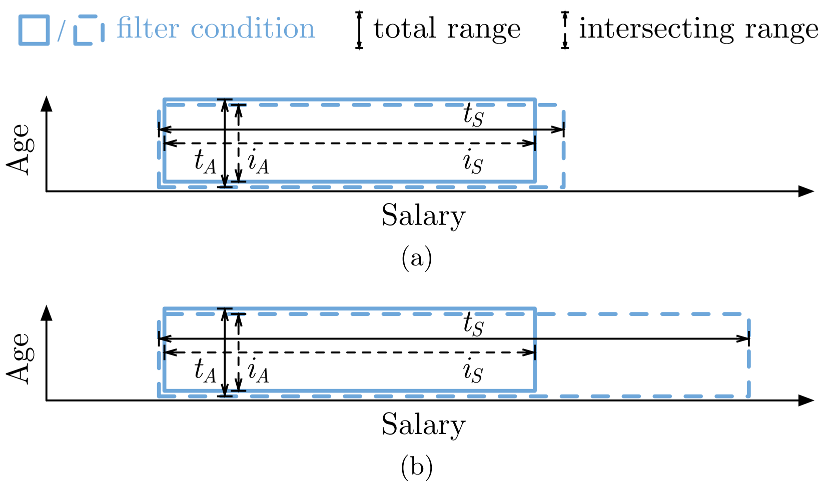


Figure 3.6.: Example scenarios depicting the evaluation of similarity between filter conditions. In the top scenario the two conditions are similar (a), in the bottom scenario they are not (b).

previous answer. In that way analysts can repeat their queries and receive an answer without incurring any budget cost. For benign analysts there should be no reason to require the same true count with fresh noise as an attacker does. As comparing filter conditions does not involve any user records, QSD does not reveal any private information and thus it is safe to return to the analyst any known data it has access to, including previous answers.

### 3.3.2. Cheater Detection

Given the additional restrictions of QSD, attackers cannot attack using the same query, i.e., the same filter condition, multiple times. It is further not possible to only change filters in a technical rather than a practical sense, e.g., add few cents to the range on salary or a few seconds to age. Instead, attackers must use multiple significantly different filters, where at least one range intersects less than 90%, i.e., differs by more than 10%, each time. Nevertheless, there are still attacks possible. However, these require more common or observed knowledge.

We continue in our example of the adversarial analyst trying to obtain information about the HIV status of one of her colleagues. In the following attack scenario we assume she possesses the additional knowledge that there is a gap in salary between the highest paid employee and all other employees.

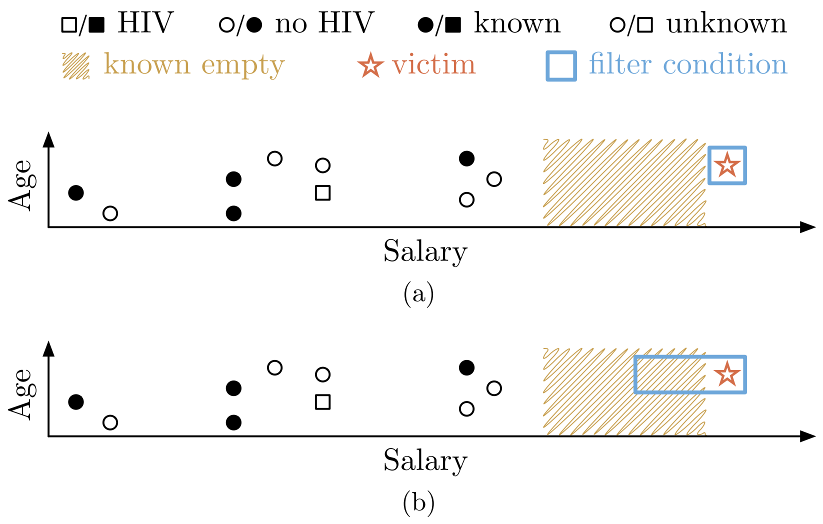


Figure 3.7.: Example of attack with different filter conditions in order to circumvent QSD. For simplicity filter conditions again do not indicate their implicit “no HIV” setting. The first query uses the same filter condition as before (a). However, the following queries extend their filters into the region that is known to be empty (b).

**Attack.** To circumvent QSD, the analyst requires additional knowledge. Figure 3.7 depicts this scenario, where the knowledge of a salary gap is marked by the shaded area. Knowing that there cannot be any employees with a salary value in the gap area allows the analyst to extend queries into that area multiple times without counting anything but the victim. The figure shows two of the queries, which both solely count the victim although they differ in the salary range by more than 10%. Given the size of the empty area multiple such queries are possible.

This attack uses observed or common knowledge to circumvent privacy protection. While observational capabilities of attackers are likely limited, they can be sufficient for an attack. Likewise it is often easy to obtain some kind of common knowledge, e.g., no Uber pickup happens in the ocean and no living human being is older than 150 years. It is thereby usually easier to obtain negative conditions, i.e., conditions that no record can fulfill, than positive conditions, i.e., conditions for which the exact number of matching records is known. Attackers then use the known negative conditions to extend and thus change their queries while resting assured that they do not count unknown records.

This attack further shows that attackers are able to use different queries to produce the same counts, in fact, count the exact same records. Coincidental occurrence of such event for any benign analyst seems unlikely, in particular as QSD ensures a significant difference between queries. Therefore, we consider it cheating behavior to count the same set of records with intersecting but different queries more than once. To detect such an event we propose *Cheater Detection* (CD), a stateful mechanism that actively monitors all true answers provided by the database and recognizes duplicate answers as cheating behavior. It thereby scans a history of previous answers and compares the sets of counted records. For intersecting queries the same set should only rarely occur. One way to implement this mechanism and make it scale to large numbers of queries is using hashes over sets of records. Instead of actually scanning the whole history, a hash based index indicates in constant time whether a given set of records has already been observed and thus whether the inspected query shows cheating behavior or not.

Upon detection of a cheating query, CD increases the query's associated budget cost to a multiple of its original value. The purpose of this increase is to eliminate any advantage the detected query might have achieved. The increase thus needs to be high enough to re-establish the conditions of DP, where cheating cannot lead to privacy loss. However, as these decisions are based on users' data, CD cannot cooperate with QBB, which is exclusively based on public data. Instead, a shadow copy of QBB runs inside CD using the updated budgets.

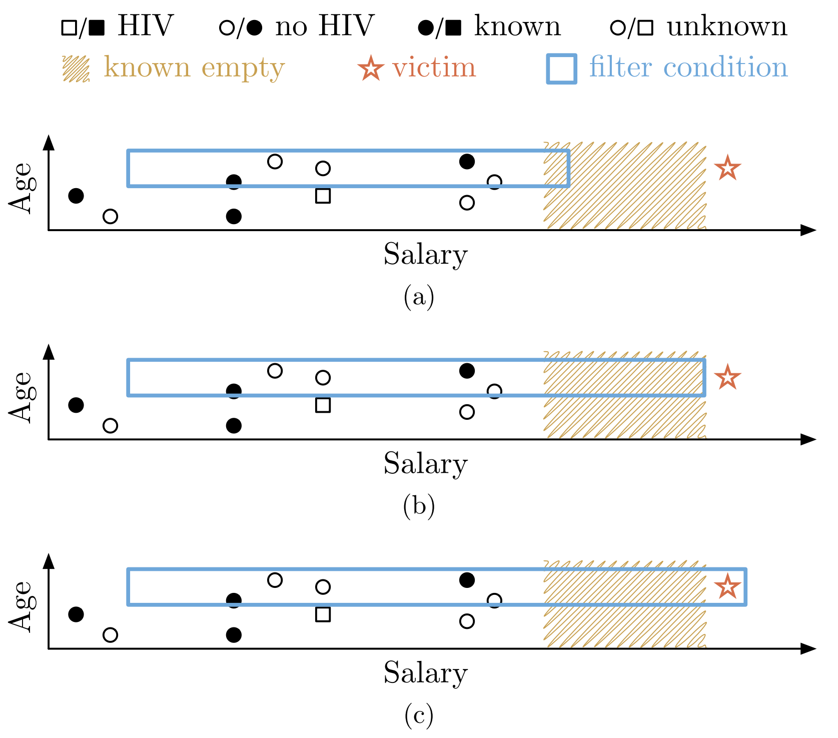


Figure 3.8.: Example of attack exploiting silent record dropping. For simplicity filter conditions again do not indicate their implicit "no HIV" setting. Attackers intentionally trigger CD by repeatedly using large filter conditions counting the same set of records (a, b). They stop right at the point where the next repetition of the same set of records will lead to record dropping while counting a different set will not. Then they extend the filter condition so it potentially matches the victim (c). If the victim indeed matches the filter the set of counted records changes and no records are dropped, otherwise all previously counted records are dropped, which is observable given a large enough initial filter.

In case CD detects that some of the updated budgets is depleted it cannot simply reject queries as that can potentially leak private information. Instead, it must suspend the analyst and disallow any subsequent queries until further inspection by an administrator in order to minimize any loss of privacy. In fact, the suspension alone already allows an attacker to answer a single true/false question about the victim. However, extended privacy loss is prevented and detected attackers could for instance be held liable to legal action.

At this point there is no way around suspending analysts. In particular, we cannot silently drop records as ProPer does, due to the possibility of the following attack.

**Attack.** For this attack we assume that the protective system employs silent record dropping instead of analyst suspension. This means that once budgets are depleted CD silently removes records from the database, repeats the query, and returns the updated count. The analyst does not get suspended. As CD is triggered based on analysts' behavior, the malicious analyst can exploit it to obtain information about a victim. To do so she uses a query that counts records next to but not including the victim (see Figure 3.8a). She intentionally repeats queries that count the same set of records, which triggers CD to increase the budget cost during each such repetition. She then stops this procedure right at the point where the next repetition with the same set of counted records would require silent dropping. However, in the next query she uses a filter that allows the victim to pass if it matches the attack condition (see Figure 3.8c). If the victim passes, it changes the set of counted records and silent dropping is not activated. Otherwise, silent dropping is indeed activated and given a large enough initial count the analyst is able to observe the drop and thus determine that the victim does not match the attack condition.

This attack shows that if a mechanism is based on information from the database as well as analyst behavior, the analyst can influence and exploit the mechanism's decisions to obtain information about users. Analyst observable effects of such mechanisms therefore need to be limited to suspension. We argue that it is ok to suspend analysts here as benign analysts are not cheating and thus are likely never affected by this mechanism. To allow for accidental activation of CD the administrator can configure lower budgets to be used in QBB. Budgets must be chosen low enough so QBB triggers before CD despite the queries that incurred higher budget costs. If QBB triggers first, queries are simply rejected and analysts are not suspended.

### 3.3.3. Gaussian noise

With both QSD and CD in place, attackers must use different queries with different results for an attack. If they don't comply they fail by either repeatedly obtaining previous and thus useless answers or by being detected and suspended. However, there is a way to fulfill these requirements and still be successful, which we show in the following attack.

**Attack.** To always count a unique set of records while never counting any unknown ones, the analyst uses multiple bounding boxes in each query's filter condition, as shown in Figure 3.9. Shown queries only include specifically chosen records and circumvent both QSD and CD.

Unfortunately for the attacker, QBB constrains filter conditions to a single bounding box per query. This attack is thus hypothetical for the case of a system without QBB. We did not want to skip this attack as it is very flexible in avoiding unknown records and thus avoiding uncertainty about the data analyzed by a query. The possibility of this kind of attack is our main argument for keeping the rule of a single bounding box per filter.

To attack using only single bounding box filters is challenging at this point. Attack filter conditions must allow for the victim to be included, must include at least one unique known record, and must not include any unknown record. In order to fulfill all these requirements in multiple different queries an attacker needs a lot of knowledge about the data in close vicinity of a victim, as the following attack shows.

**Attack.** At this point, the success of an attack depends on the specific context, i.e., the state of the data and the attacker's knowledge about that state. The analyst must be able to create multiple attack queries for a victim using only a single bounding box per query. Figure 3.10 shows an example where the attacker can successfully formulate multiple such queries. Each of the shown queries includes the victim and a different known record in order to make the set of counted records unique per query. The most difficult requirement for these queries to work is the required additional knowledge that no unknown record is counted in any of the attack queries. This knowledge of empty areas is indicated through orange shading in the diagrams. One can see from the figure how the attacker needs to know records and empty areas in close vicinity of the victim in order to attack.

This attack only works in cases where attackers possess the required knowledge. Nevertheless, attackers know when that is the case, i.e., identify victim records

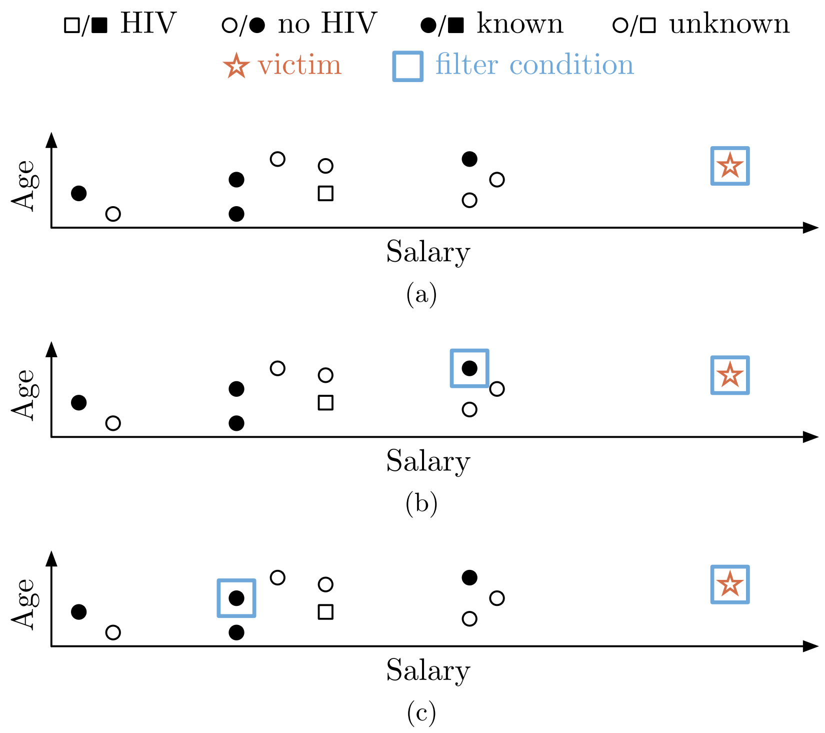


Figure 3.9.: Example of hypothetical attack using queries counting different sets of records in order to circumvent both QSD and CD. The three diagrams show three different queries, each counting a different set of records. Filter conditions on these queries are chosen to preclude any uncertainty about the resulting counts by not counting unknown records. Furthermore, each query includes at least one different record, which leads to unique sets of records being counted by all queries and CD not being triggered. For simplicity filter conditions do not show their implicit "no HIV" setting.

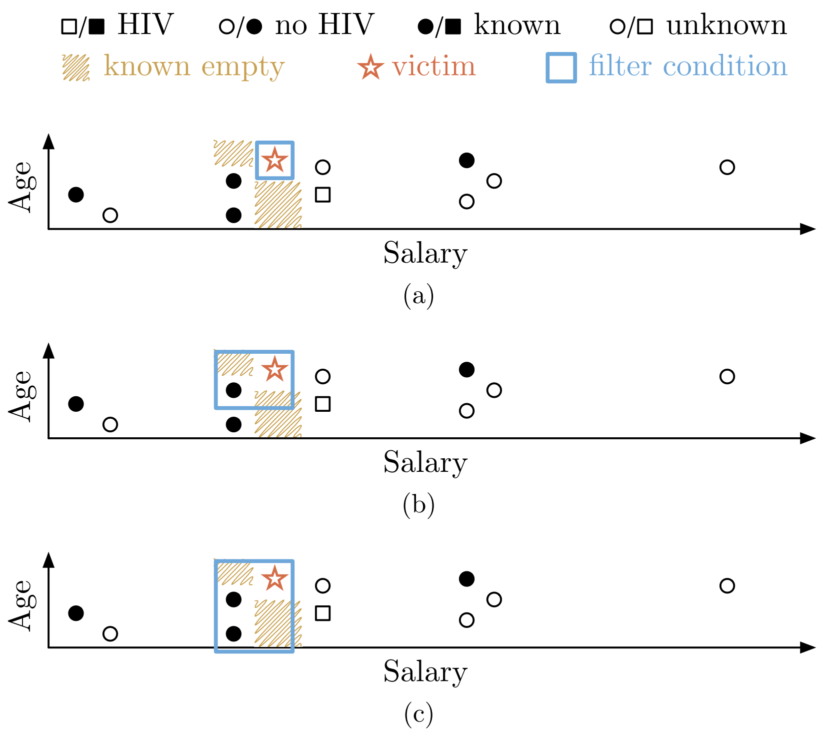


Figure 3.10.: Example of rare but possible attack where the attacker knows a lot about the data in close vicinity of a victim. In particular, the three diagrams show how the attacker is able to create three queries with a single bounding box filter each. All three filters thereby only include records known to the attacker and the victim. Again, for simplicity filter conditions do not show their implicit "no HIV" setting.

for which their knowledge is sufficient. Thus, there is the danger that attackers actively look for victims where they deem their chances of success high.

To make the success of attacks less predictable we explore the use of Gaussian instead of Laplacian noise, as would, for instance, be used on a stand-alone QBB system (see section 3.2). Laplacian noise is able to always provide a fixed bound on privacy loss. In contrast, Gaussian noise fails to do so in some cases, but provides a better bound in all other cases. Gaussian noise is thus able to provide similar protection at a lower overall level of noise. The lower noise increases utility of the system, while the random probability of failure minimizes predictability for attackers. Failures are random as it is impossible to know if and when the Gaussian distribution fails to provide adequate levels of noise. Therefore, attackers can no longer anticipate the likelihood of success of an attack, which prevents them from targeting selected weak victims.

Enough knowledge in the context of Laplacian noise allows attackers to increase the guaranteed value of epsilon in any attack as shown in Figure 3.11 from, e.g., $\varepsilon = 1.39$ to $\varepsilon = 2.78$. The former value thereby allows 80% attack confidence while doubling it results in 94% confidence. Such fixed worst case confidence cannot be provided by Gaussian noise. However, one can find Gaussian distributions that fail such upper limit on confidence only in a configurable random amount of cases. In the example we see one distribution configured for 0.5% miss rate at the level of 80% confidence and its counterpart with twice the budget, which misses in 5% of cases. As these cases are selected at random attackers cannot anticipate a fixed confidence for an attack, but only how close they are to either the dashed or the solid Gaussian distribution, which means some chance of success out of $[0.5\%, 5\%]$.

Gaussian noise allows for a different trade-off between budget, noise, and attack confidence. When a maximum bound on attack confidence is desired it allows to trade a small amount of successful attacks for lower noise and larger budgets, while restraining predictability by choosing successful attacks at random. The amount of successful attacks varies depending on the configuration of the system, e.g., where any attacker has a 0.5% chance to succeed an attacker with full knowledge of users' data gets a 5% chance to succeed. However, reaching these 5% with only partial information is hard as our restrictions impair creation of suitable attack queries. At the same time opening the system to this potential amount of successful attacks allows us to increase budgets for counting queries by another 1.8 times to a total of 3.6 times the original budget.

In summary Gaussian noise as we apply it in the *Noise Adder* (NA) (see Figure 3.13) improves utility for analysts and impedes predictability for attackers at

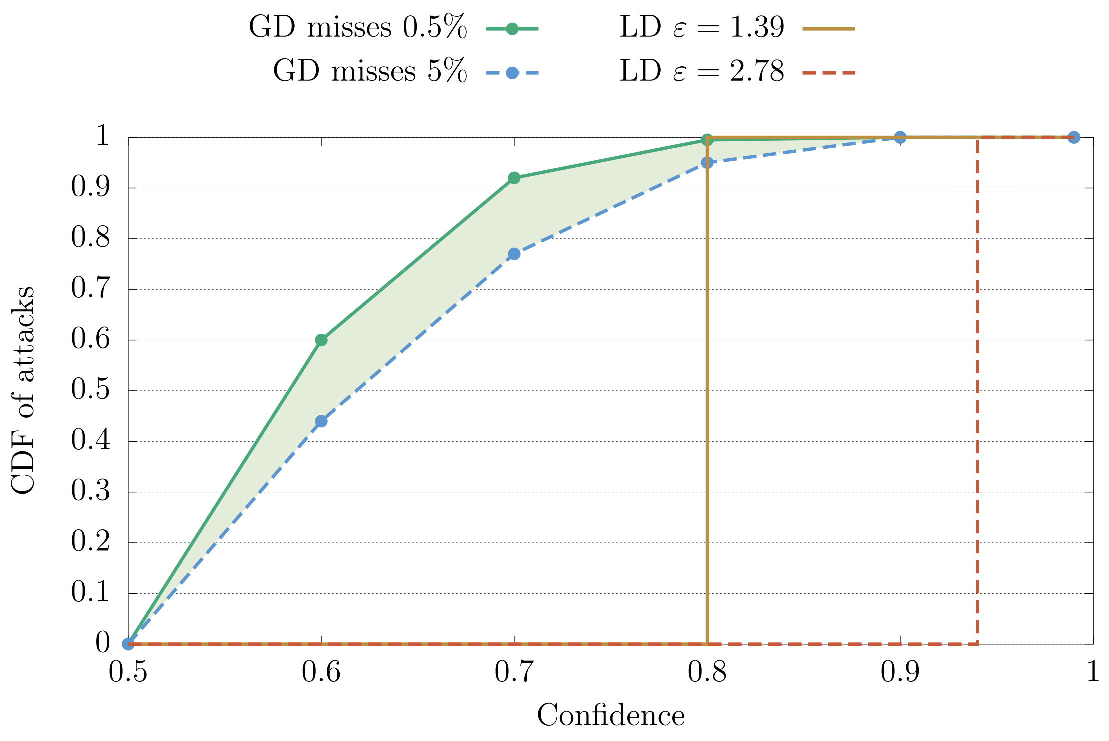


Figure 3.11.: CDF of attack confidence for different noise distributions. Distributions try to provide at most 80% attack confidence for a given budget over queries. Before doubling the $\varepsilon$-budget for chosen *Laplacian distribution* (LD) no attack achieves above 80% confidence, afterwards all attacks do. In contrast, before doubling the budget for chosen *Gaussian distribution* (GD) 0.5% of randomly selected attacks achieve above 80% confidence, afterwards 5% do.

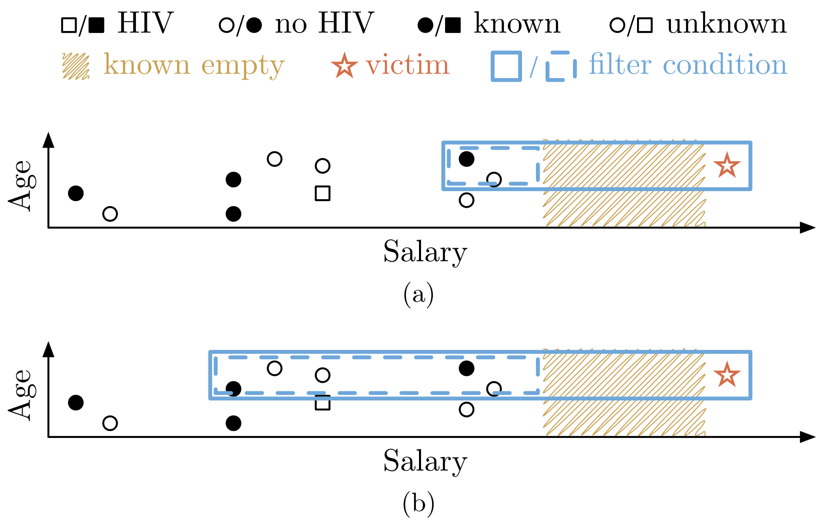


Figure 3.12.: Example of advanced attack combining pairs of queries. As before, filter conditions do not indicate their implicit "no HIV" setting. Each diagram shows a single step in the attack process, where each uses a combination of two attack queries.

the cost of a small probability that carefully crafted attacks succeed. One might now say that even if the chance of success is only 0.5%, it might hit the important few users in the database. However that would be wrong, as the Gaussian mechanism chooses successful attacks at random, the only influence the users' data might have is whether attackers manage to trick QSD and CD and thus achieve the maximum possible probability of success. This maximum probability is 5% in the case where 0.5% is the best case provided by the Gaussian noise.

### 3.3.4. Remaining attacks

To increase the probability of a successful attack despite the Gaussian noise an attacker needs to try many attacks on different victims. To be able to try more attacks with limited knowledge a more general type of attack that does not require a lot of knowledge about the close vicinity of a victim is needed. The following attack represents such type.

**Attack.** In this type of attack the analyst creates pairs of queries that need to be combined. In each attack step she creates a query that includes potentially the victim and for sure one additional known record. Then she creates a query that asks for all parts of the first query she does not know about. Subtracting the answer of the second query from that of the first removes the unknown parts from the first answer. At the cost of twice the noise the analyst is thereby able to repeatedly query the victim with fresh noise. Figure 3.12 shows two such combinations of queries, where the dashed query asks for all the parts of the solid query that the analyst is unsure about. Subtracting the answer of the dashed query from the solid query removes any unknown parts from the equation. Only the victim remains.

This attack is the reason we cannot do better than doubling the budget for counting queries. For any query that intersects the victim we have to assume there is a single other query that can ask for all the unknown parts of the answer and thus eliminate all the uncertainty an attacker might have had. The uncertainty of the data can thus be removed by exactly twice the amount of queries, which leads to twice the budget.

Next to all so-far described attacks there are few additional attacks that got excluded by assumption. In particular we make two implicit assumptions throughout the chapter that prevent attacks. The first assumption we make is taken from ProPer, which is the independence of records. We assume that records are not strongly and predictably correlated, e.g., it does not happen that all employees of a certain department get a raise together. The second assumption is that the data does not follow any clear structure that an attacker can exploit, e.g., it does not happen that there is exactly one employee per each year of age with the salary being exactly twice the age in thousands. We also assume that the attackers partial knowledge is spread over the whole database and not concentrated in a single small partition. Removing these assumptions would allow an attacker to successfully attack the system and thus require the use of a global budget. However the same is true for ProPer and similar systems.

To summarize, in this section we show how weakening the assumptions of DP in order to increase budgets requires additional restrictive mechanisms to make then possible attacks hard to launch. We learn that veering off the proven theory opens Pandora's box of attacks, which are difficult to control and impair. Nevertheless, we show three mechanisms that help to prevent many attacks and make the rest hard to launch, while preserving utility for benign analysts.

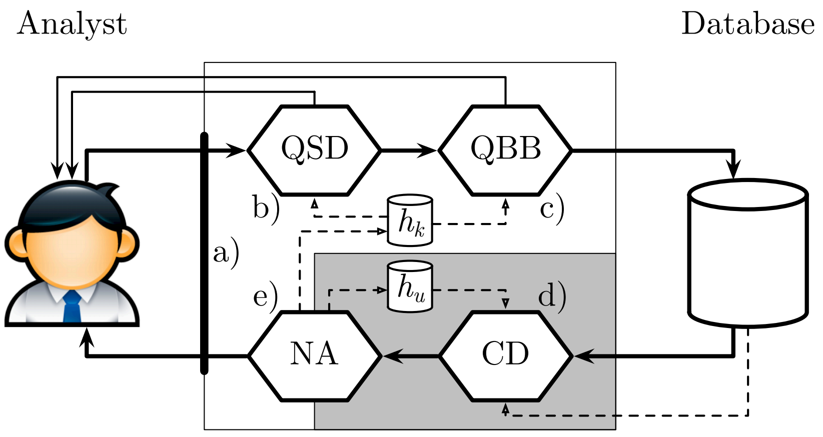


Figure 3.13.: An analyst sends a query to and receives an answer from our system (bold arrows). The query is restricted by the query interface that requires queries to be in a special format (a) and passes QSD (b) and QBB (c). Together with the true answer from the database it gets approved by CD (d). Before it is sent back to the analyst, the answer is obfuscated by the NA (e). Information from the database is only handled by components inside the shaded box, while any information outside is known to analysts. Components (b) and (c) can therefore provide feedback or previous answers to the analyst, while (d) cannot. The separation also necessitates two different query history structures, $h_k$ that contains queries and anonymized answers known to analysts, and $h_u$ that also contains the true answers from the database unknown to analysts.

```
                    ▷ Histories of queries and answers (un)known to analyst
 1: KnownHistory $h_k$
 2: UnknownHistory $h_u$
 3: function HandleQuery($q$)
                        ▷ QSD, return previous anonymized answer if possible
 4:     $h_{k,o} \leftarrow$ Overlapping($q$, $h_k$)
 5:     $q_s \leftarrow$ DetectSimilar($q$, $h_{k,o}$)
 6:     if $q_s$ then
 7:         $c_{a,s} \leftarrow$ Answer($q_s$, $h_{k,o}$)
 8:         return (similar, $q_s$, $c_{a,s}$, null)
 9:     end if
                                    ▷ QBB, reject query if budget is depleted
10:     $bAvail \leftarrow$ TrackBudget($h_{k,o}$)
11:     $bReq \leftarrow$ RequiredBudget($q$)
12:     if $bAvail < bReq$ then return (reject, $q$, null, $bAvail$)
13:     end if
                                    ▷ ENTER SECTION handling users' data
                        ▷ Obtain true answer and hash of read records from DB
14:     $(c_t, hash) \leftarrow$ DbExecute($q$)
                                        ▷ CD, suspend analyst if necessary
15:     $susp \leftarrow$ DetectCheater($q$, $c_t$, $hash$, $h_u$)
16:     if $susp$ then return (suspend, null, null, null)
17:     end if
                                            ▷ NA, anonymizes true answer
18:     $c_a \leftarrow$ AddNoise($c_t$)
                                                    ▷ Update histories
19:     Update($h_u$, $q$, $c_t$, $hash$)
                                    ▷ EXIT SECTION handling users' data
20:     Update($h_k$, $q$, $c_a$)
21:     $h_{k,o} \leftarrow$ Overlapping($q$, $h_k$)
22:     $bRemain \leftarrow$ TrackBudget($h_{k,o}$)
                                            ▷ Return anonymized answer
23:     return (new, $q$, $c_a$, $bRemain$)
24: end function
```

Figure 3.14.: Query handling of our system design in pseudo code

## 3.4. The complete design

Combining all presented components in a single design leads to a defense-in-depth system with multiple modules cooperating in order to protect users' privacy. Additionally to QBB, QSD, and CD we introduced the NA, which anonymizes true answers from the database by adding Gaussian noise. Altogether we put four separate modules in front of the database to monitor, compare, approve, and anonymize queries entering as well as answers leaving the system.

Figure 3.13 shows our setup as a diagram with the path of a regular query marked with bold arrows, while Figure 3.14 shows the same path in form of pseudo code of the query handling function. As handling queries includes stateful mechanisms, we introduced two history structures $h_k$ for queries and anonymized answers known to analysts and $h_u$ with queries and true answers unknown to analysts. Confining data unknown to analysts in a separate part of the system lowers the chance of unintended leakage for instance in the form of side-channels derived from the limitations we enforce. Components outside that part cannot cause any loss of privacy. Neither their decisions when to enforce limitations nor the additional feedback they give to analysts can possibly include any information about users' data, as by design they cannot look at such data. In Figure 3.13 the shaded area indicates the part where data unknown to analysts is used, while comments indicate the same for the pseudo code in Figure 3.14.

Before a query enters the area of unknown information it has to pass and be approved by multiple other modules. First a query passes through the query interface (Figure 3.13a). At that point it conforms to all syntactic restrictions and the system starts to handle it (Figure 3.14 Line 3). The second component is the QSD (Figure 3.13b), which obtains all previous queries overlapping the new query, i.e., with non-disjoint filter conditions, from $h_k$ and checks these overlapping queries for similar ones (Figure 3.14 Line 5). If a similar query is found it is returned together with its previous anonymized answer and the new query is dropped. Otherwise, the QBB (Figure 3.13c) determines the budget available for the query to run (Figure 3.14 Line 10), compares it with the budget required by the query, and rejects the query if it cannot fit. The rejection includes information about how much budget is still available so analysts can adjust their behavior and pose an adapted query. Once a query got approved by QBB it executes on the database, which returns the true answer and the corresponding hash of the consumed set of records to the system. At this point the query, the answer, and the hash enter the confined area where they are inspected (Figure 3.14 Line 15) by the CD (Figure 3.13d), and its own shadow budget tracking. If there is enough budget, query and answer are approved for anonymized release, which requires the NA (Figure 3.13e) to anonymize the answer (Figure 3.14 Line 18) and finally return it to the analyst.

## 3.5. Case study: taxi rides in New York City

In the previous section we put all parts together and presented our complete system design. What remains is the question of how hard these hard-to-launch attacks are, which the system allows in return for increased utility. Section 3.3 provided a general intuition about how attackers can create queries to exploit our system design and circumvent its restrictions in order to attack users' privacy. This section evaluates these intuitions in a case study on real data to empirically show that attacks are indeed hard to launch in practice.

We study the difficulties an attacker faces in two separate analyses. First, we look at how much the uncertainty about the users' data influences the attackers confidence before we secondly look at the required additional knowledge for attacks that work despite such uncertainty. Both scenarios also have a minor aspect to them. From the first scenario we derive that attackers without any knowledge of any of the users' data are not able to learn anything. The second scenario further shows the size of the data partition for which a single attack attempt consumes budget. Lower budgets impair further attacks and thus attempting to attack a record increases protection for records in its vicinity, an unexpected case of interdependency.

To show these cases we use taxi rides from the NYC taxi dataset from 2013 [26, 82]. For the following analyses we use all rides of January 2013, a set of around 14 million rides. We assume our system to protect single rides with a separate budget each. In other words, each ride represents a single user's record. We further assume a configuration of the system with a minimum noise of standard deviation $sd = 3.0$ and the goal to limit attack confidence to at most 80%. As the system is not perfect we need to determine an interval of accepted failure rates, for which we choose $[0.5\%, 5\%]$. Given these parameters the system can provide a budget of four counting queries for the minimum noise.

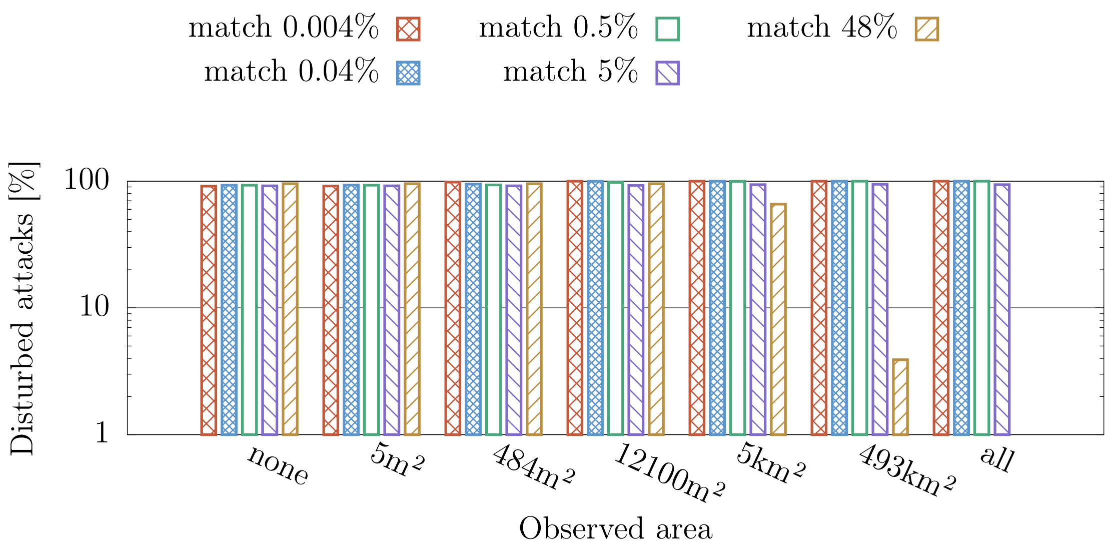
We investigate different combinations of context parameters to cover a large space of potential attack scenarios. The parameters we consider are the fraction of rides fully known to attackers, the area and time intervals in and during which they are able to observe taxi rides, and the fraction of taxi rides that share the condition of interest with the victim. This condition of interest is the information attackers want to know. We use the amount of tip, e.g., was the amount larger than \$5, for all our analyses. This choice is made with an attack in mind that actually happened for the very dataset we use [115]. We evaluate a variety of different tip amounts to show cases where fractions of rides of different orders of magnitude share the targeted condition. Similarly we vary the size of the area and the length of the time interval in and during which attackers observe taxi rides happening, but do not know the full set of data values for those rides. In particular they cannot observe the amount of tip given in those rides.

For the fraction of rides attackers fully know we assume two different scenarios. In the first scenario we try to model a single worst case attacker. We determine such attacker to be someone similar to the largest taxi operator in NYC, who operates up to 8% of all NYC taxis [118]. We thus choose a random set of 8% of all taxis for this case. Beyond that we model a scenario where the majority of taxi operators in NYC collaborate in attacks. In our case that means we randomly choose 56% of all taxis to be part of the controlled set in this case. During attacks all rides in the set of controlled taxis are considered fully known to attackers.

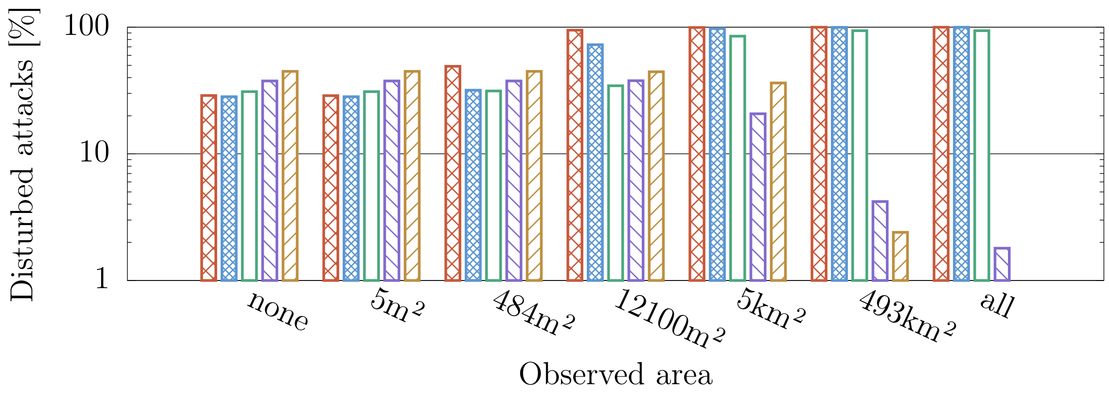
Next to these varying parameters we use the same set of randomly chosen 1000 rides as victims across both studied scenarios. We assume that attackers do not know anything about the victims except the pickup location and time, which in combination form an isolating filter for each victim, i.e., a filter condition that only contains a victim and no other ride. For simplicity we do not consider drop-off location and time in any of our investigations, however, we strongly suspect that any insight gained from the use of pickup location and times similarly applies to drop-off location and times. In general, it also applies to many other types of data columns. We merely use these particular columns as examples of observable data, which is usually used to create isolating filters. In contrast, the amount of tip, which we consider to be of interest to attackers, is the typical unknown and unobservable type of data.

### 3.5.1. Unknown records lower confidence

Our first analytic scenario evaluates the degree to which unknown records disturb a type of attack that attempts to minimize the effects of such records but cannot fully prevent them. The evaluation shows that without extensive observational capabilities even for the attacker that knows 56% of all rides more than 20% of attack attempts are disturbed for interesting conditions, i.e., conditions that match only few rides. All attackers confidence thus is below 80% for this type of attack and all attacks fail. From that result we derive that attackers without any



(a)



(b)

Figure 3.15.: Percentage of attacks disturbed by unknown records for attackers knowing 8% (a) and 56% (b) of taxi rides respectively. For each percentage the same 1000 victims were separately attacked and outcomes aggregated.

knowledge of users' data have practically zero chance to successfully attack any victim.

The analyzed type of attack is based on the one previously shown in Figure 3.10 in subsection 3.3.3. It uses queries with single bounding box filter conditions that each include a unique known record, which must match the condition of interest, e.g., tip amount is larger than $5. Attackers use their set of fully known records to fulfill this requirement. To minimize the likelihood of unknown records being counted in the attack queries, attackers use records with rides that happen either shortly before or shortly after the victim ride. As the system allows for four queries at minimum noise they need to use four such records.

Additionally, attackers exploit their observed knowledge, i.e., time and location of rides happening in a restricted area and time interval around the victim. Attackers do not use any attack queries that have the potential to count any of the records of the observed rides. As attackers do not know whether these records actually match the query condition or not, this is an overestimating measure that can remove working attack queries. In contrast to the attack described in Figure 3.10 we assume here that attackers do use queries for which they lack some knowledge of empty areas. If the finally chosen four queries do not count any unknown records, the attack achieves the highest possible probability to succeed, which is 5% for the assumed system configuration. However if any of these queries counts at least a single unknown record, the final outcome of the attack is disturbed and thus of little use.

For each combination of observable area and condition of interest we attacked the taxi ride dataset for all 1000 victim records and obtained the percentage of attacks that were disturbed by unknown records. Figure 3.15 shows these percentages for the different combinations of considered parameters. The graph above (Figure 3.15a) shows the numbers for the single worst case attacker, who knows 8% of all taxi rides. We observe that for any interesting conditions (the three left bars in each group) the amount is always close to 100%. Only for a common condition and for a powerful attacker, who can observe a very large area (the rightmost bar of the whole graph), disturbance decreases to around 0%. The graph below (Figure 3.15b) shows the same percentages for the collaboration case where more than half of NYC's taxi operators work together and together know more than every second taxi ride. Despite this impressive amount of knowledge the graph shows that such attackers still have to put up with more than 20% disturbance for interesting conditions. In practice this means that unknown records indeed increase uncertainty and lower the confidence of attackers, which therefore cannot use attacks where they lack some knowledge about empty areas.

### 3.5.2. Prerequisites for successful attacks

The first analysis shows that attackers need to know about the absence of unknown records in any attack query. Following this direction the second analysis investigates the amount of observation required to possess such knowledge. We show that for the single worst case attacker extensive amounts of observation are necessary to successfully attack even 1 in 5000 records let alone reach the best possible and attack 1 in 2000. Nevertheless, we also show that if database administrators require higher security, they can raise the minimum allowed noise in the system's configuration and attacks automatically become harder to launch as more queries are necessary and thus more data needs to be observed.

We assume the same attack as before however this time attackers only use attack queries for which they know that no unknown record can be counted. Therefore the attack now exactly resembles the example shown in Figure 3.10 in subsection 3.3.3. Observation is thereby the only means of attackers to obtain the required knowledge about empty areas. The attacker needs to observe the whole partition of data included in each attack queries' filter condition. Only queries where there are no unknown rides observed for their partition might then be used in the attack. To successfully attack 1 in 5000 records attackers need to find three attack queries, while for attacking 1 in 2000 records they need four.

Figure 3.16 shows our results for different scenarios. The x-axis of each graph shows the size of the area attackers must observe, while the y-axis details for how long they need to observe it. Up to five different lines are presented in each graph, one for each fraction of taxi rides matching the attack condition. Lines represent the amount of observation necessary to successfully attack a randomly chosen ride with a certain probability. A single line thereby represents all combinations of area and time that provide the same chance of success. The different graphs in the figure each show results for a different configuration. The upper three graphs are for the single worst case attacker controlling 8% the lower three for the collaborating attackers controlling 56% of taxi rides. The four on the left are for the currently assumed minimum standard deviation of the Gaussian noise of $sd = 3.0$, while the two graphs on the right show the effect of increasing said minimum noise to $sd = 5.0$ while keeping the other parameters constant with respect to the two middle graphs.

Graph 3.16a, for example, shows the amounts of observation needed by attackers who know 8% of all records in a system configured with a minimum standard deviation of the Gaussian noise of $sd = 3.0$. The leftmost point in that graph indicates that such attackers need to observe 484m$^2$ for 23.2 days in order to be

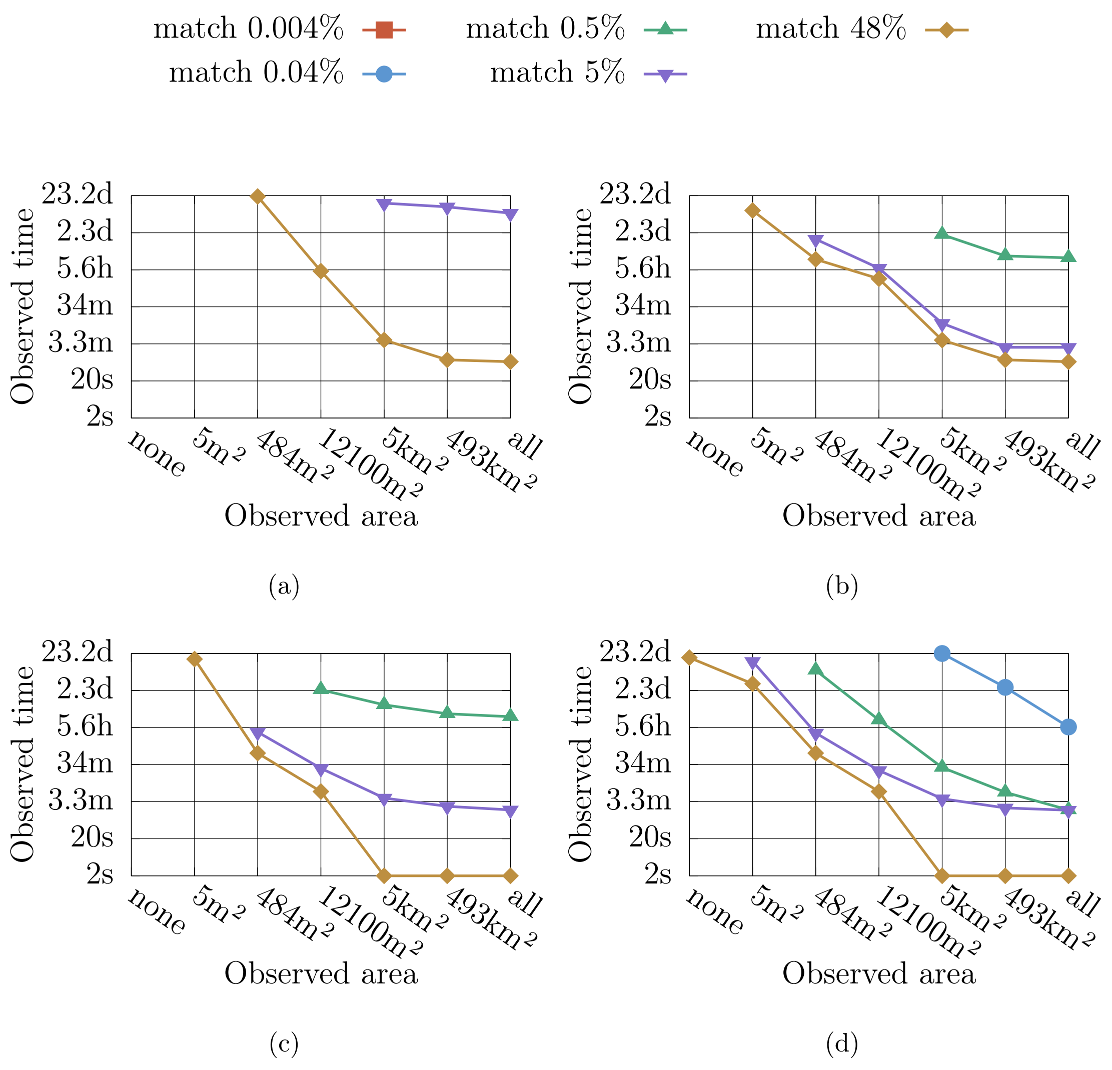
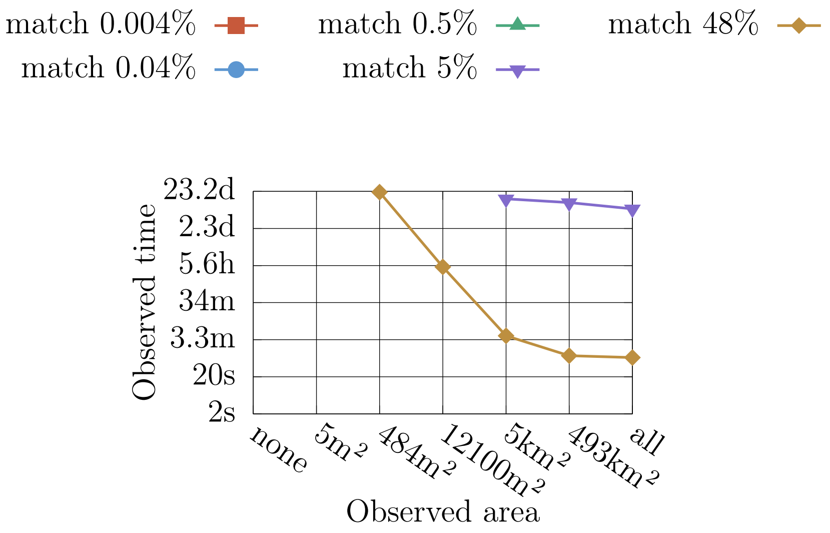
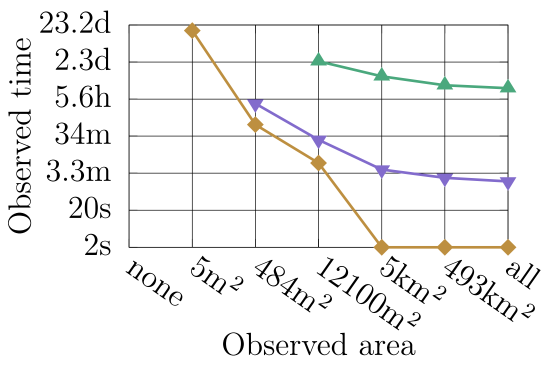


Figure 3.16.: Observational capabilities required to attack a random taxi ride. The top row (a, b) details numbers for attackers that know 8% of all rides, the bottom row (c, d) for those that know 56% of all rides. The two on the left (a, c) present numbers to successfully attack 1 in 2000 rides and the two on the right (b, d) for 1 in 5000 rides, all four for the minimum configured standard deviation of $sd = 3.0$.



(a)



(b)

Figure 3.17.: Observational capabilities required when more noise is added. The top graph (a) details numbers for attackers that know 8% of all rides, the bottom graph (b) for those that know 56% of all rides. An increased minimum standard deviation of $sd = 5.0$ improves protection and requires more observation to succeed on 1 in 5000 rides compared to a case of $sd = 3.0$ (compare Figures 3.16b and d).

able to successfully attack 1 in 2000 records with a condition that matches almost every second record. The following points on the shown line detail that attackers can likewise observe 12100m$^2$ for only 5.6 hours or 5km$^2$ for 3.3 minutes. For any condition matching less than 5% of rides there is no line present in the graph. These attacks failed due to insufficient knowledge.

Going to the right to graph 3.16b we observe the curve for attack conditions that match 5% of rides has dropped close to the line for 48%. This means that these attacks have become easier to conduct as less observation is necessary. This observation is in accordance with the fact that the middle graphs show the simpler attack that only works on 1 in 5000 victims. Going down to graph 3.16d shows the same effect of dropping lines. However, in this case it is not the attack that has become easier but the knowledge of the attacker has increased as the lower three graphs represent the case of the collaborating attackers.

The difference between the middle and the right graphs is the minimum noise allowed by the system. For the right graphs, e.g., graph 3.17b, the minimum noise of the system is increased from $sd = 3.0$ to $sd = 5.0$, which means more queries need to be used in a comparable attack. Finding additional attack queries is hard, which is why lines presented in the right graphs have risen compared to the middle graphs. In the chosen setting they actually rise to a comparable level with the left graphs, which is coincidental and has no relevant meaning.

Taking all gained insights together we learn that attackers need a lot of observed data for a large partition of the data around any victim, which means that attacks are indeed hard to launch in practice.

## 3.6. Related work

Private data analytics techniques, also called disclosure control techniques, can be roughly categorized into three groups, (1) those that perturb the original data, (2) those that perturb the output of an analytics system, and (3) those that filter, limit, and modify the queries before execution. The latter group includes techniques that suppress some outputs entirely.

Our system design works on raw unmodified data to avoid the potential biases of data perturbation [2]. Thus, we only discuss k-anonymity [109, 110] and pseudonymization [93] of the first group of systems [32, 66, 75, 94, 108]. The second group contains DP [44], research wise the golden standard of privacy protection systems. However, for our system design we took most ideas from the third group.

In the following we discuss the techniques that influenced our system design as well as the popular systems in research and industry.

### 3.6.1. The most recent and most known

The related work section of [76] provides a good overview over different kinds of techniques. It includes well-known original data perturbation techniques in the line of k-anonymity [73, 76, 88, 109, 110], but does not yet cover the more recent output perturbation techniques in the line of DP [16, 40–42, 44, 47, 81, 90]. An extensive overview of DP is provided in [46].

Despite their success in protecting privacy, both k-anonymity and DP based techniques are often criticized for the resulting loss in analytical utility [7, 17, 38, 61, 62, 74]. Either they provide good privacy but low utility, or their utility is acceptable but good privacy cannot be guaranteed any longer.

DP's application is further shown to be problematic considering specific real-world scenarios, e.g., social networks [58, 68]. This thesis only considers simpler scenarios where these problems do not occur. Otherwise, presented solutions are as susceptible to these problems as most other DP systems.

### 3.6.2. The usually applied

Pseudonymization [93] is one of—if not the—most applied ad-hoc technique for privacy protection. The technique is simple. It replaces any directly identifying data values, e.g., first and last names, with randomly chosen pseudonyms. Consistent use of the same pseudonyms for the same data values preserves inter record dependencies and correlations. Other data fields remain untouched, which allows for high data utility and increased usability.

Given its popularity, many data privacy laws and regulations require pseudonymization. Examples are HIPAA (Health Insurance Portability and Accountability Act) [27] and GDPR (General Data Protection Regulation) [25]. The first regulates the protection of medical data, i.e., patients' health records, and has lead to wide adoption of pseudonymization in respective systems (e.g., see [89, 97]). The second has come into effect only recently but governs any and all data exchange or processing in the European Union. For instance, it requires online services to apply privacy protections, e.g., to pseudonymize server logs [54].

Despite its wide adoption, pseudonymization has severe flaws. Although specific data values might not be directly identifying, they might still be unique to a specific data record or user account. Knowledge of such values allows to identify specific records and de-anonymize the data [111–113, 116]. Consider, for example, the dataset of NYC taxi rides [26, 82]. The dataset does not contain any directly identifying data values for passengers. Nevertheless, paparazzi with pictures of celebrities entering taxis were able to identify corresponding trip records via the unique combination of location and time. This led to the public exposure of celebrities' tipping behavior [115].

### 3.6.3. Related techniques of disclosure control

In [35] Denning and Schlörer give an overview over related research. Among others they cover systematic and random output rounding, query modification, query overlap control, and query auditing. These techniques anticipate the design of our system. The authors conclude that only a combination will be able to provide high security paired with low information loss, i.e., high utility.

At the same time Sichermann et al. [104], as well as later Dinur and Nissim [36], conclude that a general solution to the disclosure problem will have potentially infinite complexity and require large amounts of perturbation, i.e., noise. Although DP circumvents the complexity by making an elegant worst-case assumption, the high noise and the resulting low utility is often unacceptable. We do not make this worst-case assumption, but due to the resulting complexity we can only provide probabilistic intuitions with respect to known attacks.

Particular ideas for designing our system were taken from the following related work. There are many different topics that our system design relates to and many different techniques that influenced it. However, to our knowledge, we are the first to combine multiple of the presented individual techniques in the way that Denning and Schlörer once suggested.

**Output perturbation** Different forms of output rounding [1, 3, 28, 53, 87, 92, 99–101] are the predecessors to noise addition techniques [9, 15, 36, 39, 102, 114], which finally led to differentially private noise [40]. We use Gaussian noise to make the success of attacks less predictable as we detail in subsection 3.3.3. Gaussian noise is only able to provide an $(\varepsilon, \delta)$-privacy guarantee [43]. However, we show that despite the possibility of low Gaussian noise values an attacker requires extensive additional knowledge to successfully attack.

**Query modification** Modifying the input query to change the output in certain ways can be helpful in different scenarios. Enforcing complex access policies [107], integrity constraints [106], or lower answer granularity bounds [77] are three examples in the database security area. We modify the input query if it is similar, i.e., too close, to some previous query. In those cases we change the query to the previous one and return the previous answer.

**Query overlap control** Controlling and disallowing queries that overlap previous queries has also been investigated in different cases in the past [30, 31, 37]. Different techniques have been introduced since and even ProPer [50] can be viewed as a kind of overlap control, restricting the amount of queries overlapping on each individual user. We pick up on this idea and restrict the amount of overlap on top of each query bounding box.

**Query auditing** Inspecting the queries of an analyst to determine the potential of an attack and decide whether the next query should be allowed or not requires the auditor to recognize attack patterns. Among others, [23, 24, 37, 67, 69] present systems that can identify attacks for specific kinds of queries. In [21] Chin then concludes that for a combination of different kinds of queries complexity is a problem. Therefore, we rely on the administrator to determine whether a suspended analyst actually attacked or not. Kenthapadi et al. [67] identify the problem of potential information leakage through the banning itself. We believe this risk to be an acceptable trade-off for an increased amount of queries.

### 3.6.4. Tracker attacks

The so called "tracker attack" [33] is the basis for many attacks. All the attacks we investigated in the context of our system design are based on some version of a tracker. Trackers are often simple to find [34, 105], and despite the fact that the then-assumed system model was different and more general than ours, the attacks we investigated are still special versions of the "individual tracker" attacks.

In fact, the simplicity of finding "general" trackers led to our decision to restrict the potential query space further. Not allowing logical "OR" in queries together with demanding contiguous ranges, basically forces any attacker to use individual trackers of which he needs to find a different one for each record or user of interest. The idea of this restriction we took from [35], while the necessity of an isolating query we got from [33].

## 3.7. Conclusion

This chapter represents our second take at a system design that tries to improve usability of a private data analytics system at the cost of guarantees. Nevertheless, there is still a great deal to be learned before we can conclude that this is the way to achieve the usability analysts desire. At this point we envision three major directions for future work: to get a better understanding of usability, to formally verify the system, and to gain more practical experience.

As a first step of formally verifying our system we proof QBB correct with respect to DP in chapter 4. Beyond that, we expect it to be challenging to provide formal guarantees for mechanisms that allow hard-to-launch attacks.

Both QSD (subsection 3.3.1) and Gaussian Noise (subsection 3.3.3) can be found in adapted versions in the Diffix system [55].

# Part II.

# Returning to Differential Privacy

# 4. UniTraX: protecting data privacy with discoverable biases

Chapter 3 introduced QBB, a mechanism to track budgets and protect privacy based on queries. This chapter proofs QBB correct with respect to DP. To this end, QBB's design is improved and reimplemented in a system called UniTraX. In UniTraX budgets are tracked based on queries as was the case in QBB. Additionally, UniTraX allows more kinds of queries utilizing the *Privacy Integrated Query* (PINQ) [80] framework. This chapter compares UniTraX to PINQ's global budget tracking as well as the personalized budget tracking of the *Provenance for Personalised Differential Privacy* (ProPer) [50] system. Both of these systems are proven to be differentially private.

This chapter is a result of a collaboration between Fabienne Eigner from CISPA and Saarland University, Matteo Maffei from TU Vienna, Deepak Garg from MPI-SWS, my adviser Paul Francis, and me. Without Fabienne's, Matteo's, and Deepak's expertise the formal model and the proofs that UniTraX maintains DP budgets would not have been possible.

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 [50], that tracks per-user budgets.[1] 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 chapter 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 are included in

[1]Other DP systems also permit per-user or per-field *initial* budgets [5, 65]. 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** |

Figure 4.1.: System comparison

answers and therefore avoid unknown dataset bias. (See section 4.1 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.

The contributions of this chapter are twofold:

1. A system model and design that maintains the advantages of per-user privacy budgets, while avoiding the problem of unknown dataset bias.

2. A theoretical framework and proof that the design provides DP.

In section 4.1 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 section 4.2 and detail the theoretical framework and the proof of DP in section 4.3. In section 4.4 we discuss related work before we conclude in section 4.5.

## 4.1. 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 4.1 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≥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$\geq$60",
- "count WHERE smoker=0 AND disease=lungCancer AND initBudget$\geq$60".

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

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.

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.

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 chapter 5 for PINQ-based analyses). Thus, analysts can easily adapt to UniTraX and exploit the increased utility of per-user budgets.

## 4.2. 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 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 chapter 5.

**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*.[2] 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 [80]. An analyst performs a query 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$\geq$10 AND age$<$20 AND gender=male AND smoker=1". Next, the analyst *runs* an aggregate query like count, sum, median, 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, $\varepsilon$, that the analyst provides with the query. For lower values of $\varepsilon$, the result is more noisy, but the reduction of privacy budget is less (thus leaving more budget for future queries).

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

The novel aspect of UniTraX is how it tracks budgets. When an aggregate query with privacy parameter $\varepsilon$ is made on a selected subspace $S$, UniTraX simply records that budget $\varepsilon$ 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 $\varepsilon$'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 section 4.1.

To respect privacy budgets, it is imperative that a query with privacy parameter $\varepsilon$ does not execute on any points whose remaining budget is less than $\varepsilon$. This is enforced by query rejection, where a query is executed only if all points in the selected subspace have remaining budget at least $\varepsilon$. 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.

## 4.3. Formal description and Differential Privacy

In this section, we describe UniTraX using a formal model. We specify the DP 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 [50], which we find both elegant and natural.

### 4.3.1. 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 $\mathcal{H} = \mathcal{R} \rightarrow \mathbb{R}^{\geq 0}$. 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 $\mathcal{P} = \mathcal{R} \times \mathbb{R}^{\geq 0} \rightarrow \{\mathsf{true}, \mathsf{false}\}$. For the analyst's convenience, UniTraX allows storing a list of selected subspaces, indexed by *subspace variables* drawn from a set $\mathsf{SVar}$. UniTraX stores the association between subspace variables and subspaces in a *select table*, $T$, of type $\mathsf{SVar} \rightarrow \mathcal{P}$.

**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 Figure 4.2. Note that interactions consist of either an instruction to, or an observable output from UniTraX, or both. In detail, the interactions are:

| | | | |
|---|---|---|---|
| $a, b$ | $::=$ | $sv := sspace$ | 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 $H$ |

Figure 4.2.: Allowed interactions between analyst and UniTraX

- $sv := sspace$ 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 $\varepsilon$ 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 $\varepsilon$, 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).
- 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.[3]

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 $P \xrightarrow{Q_\varepsilon(sv)?n''} P'''$

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

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 $\alpha$ happens, then, with probability $p$, the configuration changes to $\mathbb{C}'$. $\alpha$ may be any one of:

- $\tau$: 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 Figure 4.3. These rules model the system's behavior as follows.

(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 $\varepsilon$. The execution requires all points in *sspace* to have a remaining budget of at least $\varepsilon$. A point $r$ is in *sspace* if $\mathit{sspace}(r, H(r)) = \mathsf{true}$. (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,

Update

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

Select

$$\frac{P \xrightarrow{sv:=sspace} P' \qquad sspace \in \mathcal{P}}{(P, E, H, T) \xrightarrow{\tau}_1 (P', E, H, T[sv := sspace])}$$

Read-History

$$\frac{P \xrightarrow{\text{read}?H} P'}{(P, E, H, T) \xrightarrow{H}_1 (P', E, H, T)}$$

Query

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

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

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

Reject

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

Figure 4.3.: Semantics of UniTraX

the consumption history of all points in the subspace is increased by $\varepsilon$, to record that a query with privacy parameter $\varepsilon$ 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 $\varepsilon$. 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* $\sigma$ 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 $\sigma$. Formally, we have:

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

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 (Figure 4.3) it follows that a trace $\sigma$ 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,

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

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

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

Figure 4.4.: Semantics extension for silent record dropping

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 $Q^{\mathsf{drop}}_{\varepsilon}(sv)?n$ for the analyst's program. Unlike the previously described interaction, $Q_{\varepsilon}(sv)?n$, this interaction cannot fail (be rejected). The semantics of $Q^{\mathsf{drop}}_{\varepsilon}(sv)?n$ is defined by the new rule (Query-Drop) shown in Figure 4.4. The query executes on those records in database $E$ that (1) are in subspace *sspace*, and (b) have remaining budget at least $\varepsilon$. 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 $\varepsilon$. Second, the answer of the query is returned to the analyst with probability $p$, which is determined by the same method used in (Query).

## 4.3.2. Privacy property and its formalization

UniTraX respects the initial privacy budget of every record added to the database in the sense of DP. Before explaining this property formally, we recap the standard notion of DP due to Dwork et al. [40].

**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 $\varepsilon$-

differentially private if for any two databases $D, D'$ that differ in one record and for any $V' \subseteq V$,

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

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 $\varepsilon_1$-, ..., $\varepsilon_n$-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 $\varepsilon$-differentially private query, $\varepsilon$ 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 $\vec{Q}$, and for any sequence of observations $\vec{o}$,

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

where $\eta$ is the privacy budget.

**Our privacy property.** We use the same privacy property as ProPer. This privacy property generalizes DP described above by accounting for 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 $\sigma_0$ and $\sigma_1$ that can result from the same starting configuration. Suppose that $\sigma_0$ and $\sigma_1$ 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 $\eta$ is the sum of the initial budgets of all records in which the database updates differ between $\sigma_0$ and $\sigma_1$.

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 $\sigma_0$ with

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

Figure 4.5.: Trace distance

probability $p_0$, then the probability that the analyst makes the *same* observations under database updates from $\sigma_1$ 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 $\sigma$ and $\sigma'$ differ if $\sigma$ and $\sigma'$ agree pointwise on all labels other than database updates. If $\sigma$ and $\sigma'$ differ at a label other than database update then $dist(\sigma,\sigma')$ is undefined. The formal definition is shown in Figure 4.5.

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

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 proof was worked out by my collaborators and can be found in Appendix A as well as in our technical report [85].

## 4.4. 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 [76] provides a good overview of early work in this space. Additional background information can be found in section 3.6.

Around ten years ago Dwork et al. introduced *Differential Privacy* (DP) [40], which quickly developed into a standard for private data analytics research (see [46]). In this section, we focus on research that investigates heterogeneous or personalized budgets, tracking of personalized budgets, and private analytics on dynamic datasets.

Alaggan et al. [5] 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 [65]. 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. [91] and earlier research projects [29, 59] 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 [50] is the only system that allows this. We compared extensively to ProPer in section 4.1. Our formal model in section 4.3 is also based on ProPer's formal model.

Similar to ProPer and UniTraX, Google's RAPPOR [51] provides DP guarantees based on user-provided parameters. However, 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.

Recently, DP is being increasingly applied to dynamic datasets rather than static databases. Since the first consideration of such scenarios in 2010 [45], numerous systems have emerged [18–20, 57, 95, 96, 103] 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.

## 4.5. Conclusion

This chapter 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 DP for all records. Next, chapter 5 introduces a prototype implementation of UniTraX and an evaluation of its usability in comparison to its costs. In particular, the evaluation quantifies saved privacy budgets as well as latency overheads for different realistic workloads.

# 5. Testing UniTraX's usability

Data analytics systems should be fast, accurate, and generally applicable. Private data analytics systems at the same time also need to protect the privacy of users whose data is analyzed. The golden standard are differentially private systems that provide *formal* guarantees of protection. To achieve these guarantees, such systems commonly return only numerical statistical output to analysts with additional artificial random noise added to each output. The amount of noise can be configured by analysts.

Despite being considered a golden standard, differentially private data analytics systems have a severe usability problem. At usable amounts of noise they do not allow enough queries, i.e., when probabilistic bounds on the noise added to statistical outputs satisfy analytic needs, the number of queries possible at that amount of noise is too low. I believe this to be the reason for the low adoption rate of DP in industry. Industry to date prefers *ad-hoc* mechanisms that allow unlimited queries.

One way to mitigate this problem is UniTraX, a DP system that allows more queries than previous systems without giving up on result accuracy (see chapter 4). Similar to previous differentially private systems UniTraX protects privacy by tracking privacy loss across queries. Once a certain budget of privacy has been depleted, no more queries are possible. The goal of UniTraX is to provide more queries for a given budget than other systems. It so does by tracking different budgets for distinct parts of a database's parameter space. Queries over one part then do not use the budgets of other parts. The saved budget is available for additional queries that would otherwise not be possible.

However, UniTraX's budget savings come at a cost. 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. Excessive query latency overheads can severely impair UniTraX's utility and defeat its purpose of improving the usability.

To understand UniTraX's costs and benefits, I built a prototype implementation of UniTraX on top of PINQ [80]. This chapter presents both the implementation as well as its evaluation. By carefully clubbing budgets over contiguous regions of the parameter space, I am able to save significant amounts of privacy budget with average overheads of less than 70 % over a no-privacy baseline on realistic workloads. These results indicate that UniTraX is generally applicable and works with moderate overheads across a range of different domains.

Altogether, this chapter continues to follow the proposed paths of future work from the end of section 3.7. These suggested to work on (1) a better understanding of usability, (2) a formal verification of QBB, and (3) more practical experience with it. Chapter 4 implements the second point by introducing UniTraX, an improved version of QBB, and proving it formally correct with respect to the rules of DP. This chapter continues with the first and third points. It makes the following two contributions.

1. An implementation of UniTraX that allows analysts to save privacy budgets across a range of different domains and analytic scenarios.
2. An evaluation showing that the system is able to save significant amounts of privacy budgets at average overheads of less than 70 %.

The chapter is structured as follows. I first present the implementation of UniTraX and its features in section 5.1. In section 5.2 I then detail UniTraX's evaluation. Among others I present the datasets for the evaluation in subsection 5.2.1, the evaluation's analysis sessions, i.e., analytic tasks, in subsection 5.2.2, details on the experiments in subsections 5.2.3 to 5.2.5, and the actual results of the experimental evaluation from subsection 5.2.6 onwards.

## 5.1. Implementation

I have implemented UniTraX on top of PINQ, an earlier framework for enforcing DP with a global budget for the database [80]. I briefly review relevant details of PINQ before explaining my implementation.

**PINQ review.** PINQ adds DP to the *Language Integrated Query* (LINQ) framework, a general-purpose database query framework and well-integrated declarative extension of the .NET platform. LINQ provides a unified object-oriented data access and query interface, 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 [117].

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.

PINQ provides a thin DP wrapper over LINQ. Building on LINQ, PINQ maintains a global privacy budget for the entire database. For all queries, it ensures that sufficient budget is available and that returned answers are appropriately noised. 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.

My implementation uses PINQ in an unconventional way—I initialize a new PINQ object prior to every data analysis, and use PINQ to enforce a stipulated budget. Additionally, I track budget consumption on subspaces of the parameter space across queries.

For the purpose of this thesis I consider PINQ as a black box system. I thus consider any bugs or vulnerabilities [48, 60] as orthogonal. Applicable solutions to these problems are described in the respective papers. I do not believe these problems to be fundamental as a simplified and verified implementation of PINQ exists [49]. However, in this thesis I use PINQ's original implementation as published by its author [79].

**UniTraX implementation.** My implementation currently supports only query execution with rejection. The main addition to PINQ is tracking of consumption budgets over subspaces. *In principle*, I 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, my implementation tries to cluster contiguous subspaces with identical consumption and represents them efficiently.

My 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.

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

**Typical analysis workflow.** I briefly describe the steps an analyst must follow to run an analysis on my 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.

**Hidden columns and tables.** All implementations of DP systems that I am aware of require the protected data to be numerical and the database schema to be very simple. Usually, the protected database consists of a single table where each row represents a different user's data. UniTraX requires the same data layout. However, not all datasets conform to this typically assumed layout. Two of the problems that I encountered are string columns and one-to-many relationships.

For the datasets I work with in the context of this thesis most string columns actually represent an enumerable datatype. For such columns the number of distinct string values is small and the values themselves do not leak any private information. For example, a column might indicate whether a customer paid with "cash" or "card". Such column can easily be transformed into a numerical datatype by establishing and publishing a mapping of numerical to string values. However, when a string column contains private information, e.g., the content of a comment field, values cannot be published and thus the column cannot be transformed into a useful numerical format.

A different problem are one-to-many relationships. These occur when a user can have an arbitrary number of a specific data item, e.g., transactions. To fulfill the requirement of a single row per user one can flatten the list of transactions per user and add them to the end of the row. However, in many cases the resulting high number of columns leads to extreme performance penalties.

To solve both these problems UniTraX uses hidden columns and tables. Such hidden data might not be aggregated and returned to analysts. However, analysts are able to use hidden data in query conditions and are thus able to have other aggregated values, e.g., counts, depend on the hidden data. Analysts are thus still able to obtain limited statistical information about the hidden data. At the same time UniTraX is able to prevent private information from leaking and does not suffer from any extreme performance penalties.

**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, I assume records to be timestamped on arrival; this timestamp is another column in the 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.

With UniTraX analysts are not only able to use the just described sliding window analysis but also a more flexible multi-windowed analysis. There, the analyst can have concurrent windows with varying sizes. Budgets are only used when an analysis is performed on a window. It is thus possible to allocate all budget to a single window and none to the concurrent windows. The multi-windowed analysis allows to have analyses for short time ranges run in parallel with analyses on long-term ranges. For instance, one analysis might run every ten minutes and analyze data of the previous hour, while another analysis only executes at the end of every month for all the data of the full month.

**Clean-up mechanisms.** UniTraX tracks budgets for all data points in the whole parameter space of a database. Instead of actually storing a budget amount for each datapoint, UniTraX actually stores a compressed history of budget usage on subspaces. Each new query's subspaces might thereby intersect with a number of previous subspaces and thus produce multiple new entries in the history structure. However, the more entries there are in the history the longer it takes to check whether there is enough budget left for the next query. Therefore, the history structure needs to be cleaned in regular intervals.

Cleaning UniTraX's internal history structure can be a time consuming operation. While it can be done asynchronously and thus for a single query does not add to end-to-end query latencies, it still matters for batch processing. To give administrators and analysts more control over when cleaning operations are performed, UniTraX provides two different mechanisms, re-balancing and cleaning threshold. The threshold delays cleaning until a specific amount of additional state has accumulated. It provides control over when time is spent for cleaning but does not help reduce the size of UniTraX's internal state. Re-balancing on the other hand does allow the analyst to reduce UniTraX's internal state at the cost of additional budget. The following two paragraphs detail the differences of re-balancing with an example.

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 taxi fares between \$0 and \$100. The analyst may request ten \$10 bars. As long as each bar consumes the same budget, UniTraX will optimize tracking state and merge the subspaces of these ten 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, which results in 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 some cases by more than an order of magnitude. Re-balancing thus allows analysts to trade-off overheads against budget savings.

## 5.2. Evaluation

This section presents an evaluation of the usability of my implementation of UniTraX and its performance. Of primary interest to me are the amount of saved privacy budgets and 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, I 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, I briefly describe the datasets I use and the queries I run. The queries are deliberately chosen to be simple since long-running, complex queries will mask UniTraX's relative overheads.

In the following subsections I first introduce the different datasets before I describe the queries I use throughout different analysis sessions on the datasets. I then provide details on the different experimental setups, i.e., systems, that I compare against each other, together with descriptions of my experiments' hard- and software setup. Finally, I present and interpret my experimental measurements and results.

### 5.2.1. Datasets

I use three real-world datasets from three different domains, namely the mobility, financial, and medical domains. First, working with different domains shows

UniTraX's versatility. Second, it is a piece of evidence that obtained experimental results apply in a universal sense.

For each dataset I modify its records to contain numerical data where possible. When used with UniTraX, I add an additional initial budget for each record. For the purpose of my measurements all budgets are chosen high enough so that no budgets expire.

Where transformation to numerical data is not possible, e.g., where publishing a mapping to numerical data would leak private information, I make use of hidden columns and tables (see section 5.1). The use of hidden data is explicitly indicated in the following descriptions of the different datasets. All of the following datasets are publicly available online.

**Mobility records.** The first dataset is from the mobility domain and consists of taxi rides from New York City (the NYC taxi dataset) [26, 82]. It is publicly available from the Taxi and Limousine Commission's website, which covers rides of yellow and green taxis as well as some private limousine services and contains the data of over a decade of taxi rides. Throughout this thesis I use the yellow cab rides reported for January 2013 ($\approx$14 M records; see section 3.5).

The dataset consists of a single table where each row contains the data of a single taxi ride. This includes the date and time for pickup and dropoff of a ride as well as GPS coordinates of the respective locations, trip distance, trip time, and the number of passengers in the taxi. It also provides a detailed run down of payments with separate fields on tax, surcharge, tolls, tips, the fare amount, and the grand total.

A ride is the individual protected entity for this dataset, i.e., I assume rides are independent of each other and each ride has to be protected individually. This assumption might not always hold, e.g., the same person might take the same ride every day. When the assumption does not hold the protection guarantee is lowered to the sum of budgets of all dependent records. However, the dataset does not specify any unique passenger identifier so it is on one hand impossible for us to know all rides of a single user and adequately protect them, on the other hand it also requires extensive additional knowledge on any attackers side to still attack. In this thesis taxi drivers are not considered protected individuals.

With the exception of vendor ID, rate code, store and forward flag, and payment method all fields of the taxi ride dataset are numerical. For simplicity, I remove the non-numerical fields and only use the numerical ones. In comparison to my dataset, the currently downloadable version of the dataset does not contain the trip time in seconds as a separate field. Older versions of the dataset additionally contain medallion numbers and hashes of drivers' licenses, which uniquely identify car and driver across ride records. These do not exist in the current version of the dataset and are not used in this thesis.

**Financial records.** The second dataset contains customer data of a Czech bank. The data is anonymized and got published by the PKDD'99 discovery challenge [13]. Anonymized in this case means that customers' names are removed and addresses are limited to district granularity. I am not aware of any further changes to the data. Note that this simple form of anonymization cannot defend against an attacker with additional knowledge. For example, if an attacker knows amount and date of a single transaction of a victim, it is highly likely that such attacker gets to know all financial records of this victim. The attacker only fails in case another user has an identical transaction.

The financial dataset consists of eight tables with information regarding bank accounts and clients. In detail these are the following tables.

*Account.* Provides information on when the account was created and in which district. It also specifies the frequency at which statements are issued.

*Card.* Details a card's type, issue date, and its respective disposition, i.e., a client to account relationship that the card is associated with. There are no clients with more than a single card.

*Client.* Only lists a client's birthdate and district. The client's sex is encoded in the date using a custom format. Other data has been removed during the anonymization process.

*Disposition.* A disposition is a client to account relationship. Each records specifies a client, an account, and whether the client is the account owner or only a simple user. There are no clients with more than a single account. All cards are associated to account owners, none to account users.

*District.* This table provides 15 properties for each district. Among others it specifies number of municipalities of different sizes, average salary, unemployment numbers, and crime rates for two years.

*Loan.* When an account has a loan associated, this table shows the date the loan was granted, the loaned amount, the duration, the height of monthly payments, and the loan's status. There is no account in the dataset with more than a single loan.

*Order.* The order table contains all accounts' permanent orders. Each order is associated to a single account, specifies the receiving bank and account, the amount, and the type of payment. An account can have arbitrary many orders.

*Transaction.* Similar to permanent orders, transactions capture all changes to an account's balance. Next to the account, amount, date of transaction, the bank and account the transfer came from or went to, and the balance on the account after the transaction, the table also holds information on whether the transaction went into or out of the account, the kind of operation the transaction was part of, and a characterization of its purpose. Again, an account may have an arbitrary number of transactions associated.

Most of the data in the financial dataset refers to or is associated with a bank account. Therefore, I use accounts as the protected entities in my experiments. Given the absence of clients with relations to multiple accounts all clients are naturally independent and thus fully protected for this particular dataset. Otherwise, clients with multiple accounts would only be protected with the sum of privacy budgets across all their accounts.

Both orders and transactions have many-to-one relationships to bank accounts. These relationships require the use of hidden tables. In general, cards and loans could also have many-to-one relationships. However, in the given dataset both only have one-to-one relationships. Thus, I am able to merge these tables into the main—account—table. In the end four tables remain, the account table and one hidden table each for districts, orders, and transactions.

**Medical records.** The third and final real-world dataset I use is from the medical domain. It is another dataset from the PKDD'99 discovery challenge and contains a Japanese hospital's patient data [13]. Similar to the bank dataset personal patient information was removed during anonymization. Again, anyone with access to a single laboratory report of a patient or similar data is able to de-anonymize the data. Such attacker would then have access to data considered most personal.

The dataset focuses on thrombosis as a common symptom of Collagen diseases. It consists of three tables, one with patient data, one with regular laboratories, and one with special laboratories conducted by a lab specialized in Collagen diseases. In detail these are the following tables.

*Patient.* Basic data on each patient is provided in this table. It holds the patient's sex, birthday, date and type of first admission, diagnosis, and the date when the record was added to the system.

*Standard labs.* This table holds standard hospital laboratories. Standard means that these laboratories hold results of regular examinations as done for any patient regardless of diagnosis. The table holds details on over 40 different lab values including the date of the examination. A patient may have arbitrarily many standard laboratories.

*Special labs.* These laboratories were conducted by a lab specialized in Collagen diseases. It provides an additional 11 lab values specifically related to thrombosis and Collagen diseases. Although patients could potentially have multiple special laboratories there is no patient that has more than a single special lab in the dataset.

Naturally, patients are the protected entities for this dataset. As the relationship between patients and special laboratories is one-to-one for this particular dataset I could also use the special laboratories as protected entities. However, not all patients have such a special laboratory as these were only conducted for patients where thrombosis and a resulting potential for Collagen diseases were strongly suspected. Therefore, patients are the protected entities for this dataset.

The medical records turn out to be the worst to transform as they are dirty and ill maintained. For most part the dataset looks like a digitization of manually filled documents. For many types of laboratory data a multitude of different formattings are used to describe the same value. For example, values "<5", "5-", and "5>" would all be used to indicate a value of less than five. I am thus not claiming full correctness for the data that I finally use with UniTraX. I transform the data to the best of my knowledge and ability but in some cases I am simply lacking the knowledge of a medical professional.

Given the one-to-one relationship between patients and special laboratories I am able to merge both tables. However, as not all patients have such a laboratory I must encode null values wherever the data is missing. This leads me to the use of hybrid columns, numerical columns that contain both continuous values as well as values that are part of an enumerable mapping. For example, many lab values are positive by nature. In those cases I encode null by using a value of -1. The table with standard laboratories cannot be merged with the main table and stays a separate hidden table.

### 5.2.2. Analysis sessions

To my knowledge there are no established reference analyses available for private data analytics. By reference analyses I mean query traces that are deemed representative for the types of analyses analysts would want to run against privacy protected databases. Given their absence I tried to the best of my abilities to create my own set of representative analyses. In the following I briefly describe the four sessions, i.e., sets of queries, which I use for my experiments.

**Mobility session.** The session for the taxi ride dataset is roughly patterned off of the analysis of the same dataset described in [52]. 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 $16 \times 16$ 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.

**Financial session.** For the fianancial dataset's analysis session I assume an analyst wants to investigate female clients. After obtaining counts for male and female clients, the scenario requests queries for six different histograms. Each histogram analyzes the female clients' bank accounts along a different dimension. Among others, the analysis includes a bucketization along the dimension of sum of transaction amounts. This bucketization represents a special histogram as it is formed over data from a hidden table.

Note that hidden data cannot be directly aggregated and returned, e.g., it is impossible to return the average transaction amount across all clients. In a histogram, however, the data is only used to decide whether a client should be counted or not, which is allowed.

**Medical session.** Similar to the financial session I assume that an analyst wants to know more about severe thrombosis patients of the medical dataset. First, the analysis session issues queries to obtain a histogram over the different grades of thrombosis. In the second step the session creates 15 histograms over different features of patients with the most severe diagnosis of thrombosis. There is not only a histogram over the hidden laboratory table but also one over a hidden column in this session. In that case the histogram simply counts patients that have a certain abbreviated diagnosis contained in the hidden diagnosis field (of type string).

**Data stream session.** I consider one special analysis session, a data stream session. It shows that UniTraX is not only able to provide DP protection to static but also highly dynamic streaming datasets. In a streaming dataset new data is constantly added with the same queries issued on new data once a specific amount has been added or a certain value range has been reached. The most prominent kind of value range is time ranges. The newly arriving data thereby carries increasing timestamps and the same queries are issued for different windows in time.

In traditional global budget systems like PINQ each query to a time window reduces the global budget. The budget thus depletes rapidly. UniTraX, in contrast, reduces budgets of records in distinct time windows only once. Therefore UniTraX is able to save the majority of privacy budgets compared to previous global budget systems. The saved budget can then be used on additional queries, e.g., on larger time windows.

I use the taxi rides dataset for this analysis session. To save on experiment runtime I do not add records over time. Instead the session simply queries data over different windows of time. UniTraX is oblivious to the data so it does not change any results if the data is static. On the contrary, adding data in between queries creates significant database overheads that have the potential to mask UniTraX's overheads. Database overheads are thereby mostly due to index updates. To avoid interference of database overheads I add all the data beforehand and only act as if it were added in between queries.

For the data stream session I assume that an analyst is interested in the average speed of taxis going from southern Manhattan to Times Square. I further assume that the data needs to be updated every hour, e.g., to provide traffic estimates. Therefore, the session executes queries for average trip time and trip distance for every hour of data. These queries include a conditional query that only executes the following queries if at least 25 taxi rides happened during the time window of interest.

To show UniTraX's ability to flexibly handle time windows of any size and combination, queries not only try to run for the previous hour of newly added data. If the postulated minimum number of 25 rides is not fulfilled queries repeat for larger time windows of the previous two to six hours before they give up. Note that only the first previous hour consists of newly added and thus untouched data. Records of the other hours have been queried already.

I further assume that a second analyst is interested in the monthly average trip time from John F. Kennedy airport to each of the $16 \times 16$ square areas of Manhattan, which I already used in the mobility session. The analytic scenario thus runs additional queries on all the data of the dataset at the end of the session. Queries again test for a minimum of 25 rides but do not continue for different time frames if not enough rides are available.

### 5.2.3. Experimental setups

I run the analysis sessions 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

Unless otherwise indicated UniTraX is configured to clean its internal data structures after each query. During the evaluation of different clean-up mechanisms I compare to three additional UniTraX configurations. (1) A clean-up happens only once 500 additional subspaces have been added. (2) The analyst uses re-balancing to keep the size of UniTraX's internal state small but cleaning executes after every query. And (3), both re-balancing and clean-up after 500 are applied in combination.

### 5.2.4. 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 disabled (total of 16 cores = hardware threads per machine) and 256 GB of main memory. Both systems are connected to two separate top-of-rack switches with one 1 Gbit/s connections each. The two connections per machine are bonded and use MAC based layer two routing. This was the default network configuration provided to me, which could not be reconfigured.

### 5.2.5. Software

UniTraX is based on PINQ. To be able to use PINQ as provided, I adapted to its requirements and dependencies, which reflect in the following software choices.

**OS & Database.** I use Microsoft Windows Server 2016 on both blade systems. The first system runs both UniTraX as well as the client query program. Microsoft Visual Studio Community 2017 is the only additional software installed for these tasks. The second system runs Microsoft SQL Server 2017 Developer Edition as the remote database server. To optimize database performance I put data and index files of my database onto a RAM-disk, create indexes that fit my queries, and make the database read-only.

Microsoft SQL Server further provides a performance feature called "memory-optimized tables", where all data is kept in main memory. However, for my experiments this feature resulted in a performance degradation for most queries. I thus do not use this feature for the purposes of this thesis.

To obtain comparable performance numbers I set the concurrency parameter of the database server to sequential execution. This means that single operators execute utilizing only a single thread. However, different operators might still execute concurrently. Forced sequential execution is necessary as the database uses different numbers of threads for operations on differently sized indexes. UniTraX's budget column increases the size of its database indexes and would thus lead to incomparable performance numbers when sequential execution is not enforced.

**Query library.** The three systems of my experimental setups each provide a syntactically different query interface. Thus, each of them requires its own special implementation for each query. However, ensuring the semantic equivalence of multiple separate query implementations is complex and error prone. To mitigate this problem I introduce a query library for equivalent query composition. My library allows to compose single query objects that are able to automatically provide a query implementation for each of the three systems.

**Experiment automation.** Similar to the different query interfaces, UniTraX additionally requires a different database setup. For UniTraX each dataset must include the initial budget column, which also needs to be added to each index. In order to guarantee identical database states at the beginning of each experimental run, I implemented a second library for automated experiment orchestration. With this library I am able to automatically wipe and reinitialize the database with the accordingly configured dataset. It allows me to automatically run a large number of differently configured experiments.

### 5.2.6. Relative budget consumption

Figure 5.1 presents UniTraX's budget consumption relative to PINQ's. The graph shows a CDF for each analysis session described in subsection 5.2.2. For each session the CDF ranges over all records in its respective dataset. The x-axis reports on UniTraX's budget consumption on a record divided by the corresponding consumption of PINQ. Budget consumption for a record is thereby defined as the difference between the record's available budget before and after the respective analysis session has executed.

Please note that I only report on existing records in the dataset here. The graph does not show budget use for data points that do not have a corresponding record in the dataset.

The figure provides a curve for each of my four analysis sessions, mobility, financial, medical, and streaming. The further each curve reaches into the upper left corner the more budget UniTraX saves compared to PINQ. At the 99th-percentile UniTraX uses less than 1 % of budget in the mobility session, less than 2.5 % in the financial session, less than 10.5 % in the medical session, and practically none in the streaming session. It is obvious that UniTraX is able to save significant amounts of privacy budget in comparison to a traditional global budget system.

Nevertheless, these results depend on the specific session. Different queries lead to different results and not all combinations might save as much as the presented. However, with histograms and spatial analyses I use tasks that are commonly performed by data analysts.

**PINQ partitioning.** PINQ is not a pure global budget system. To increase budget efficiency PINQ provides a partitioning operation that allows splitting the data into distinct parts, track a separate budget for each part, and at any point in time subtracting the maximum among these budgets from the original

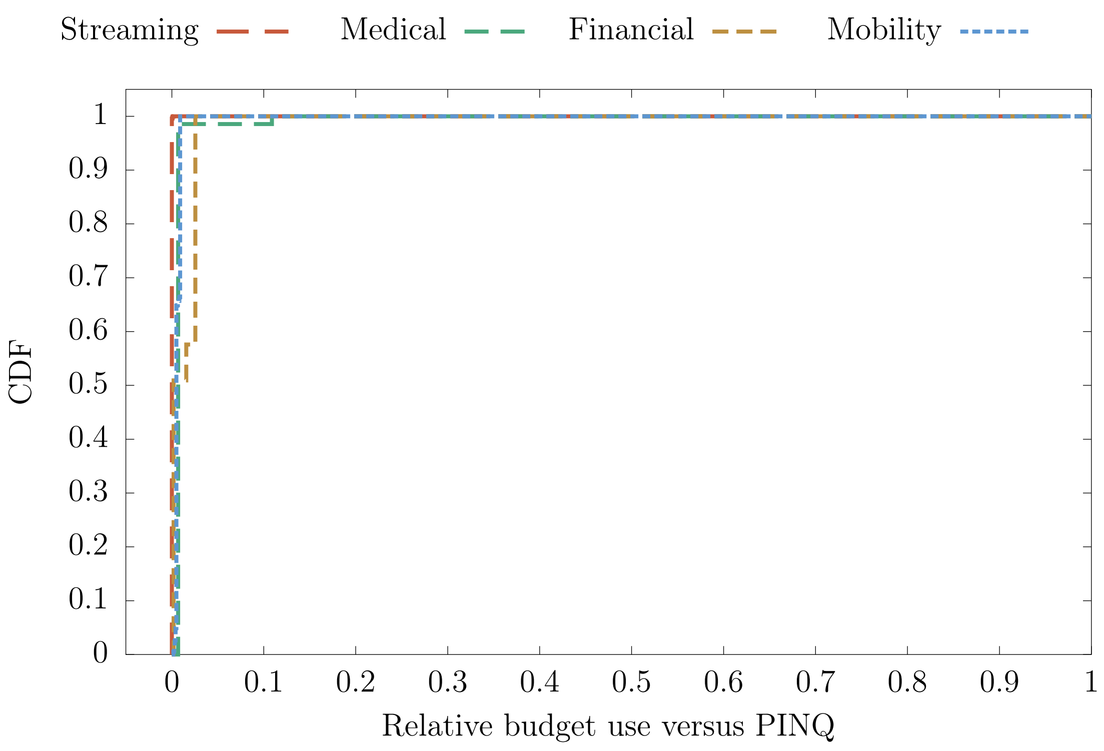


Figure 5.1.: CDF of relative budget use of UniTraX versus PINQ during different analysis sessions. At the 99th-percentile UniTraX uses less than 1 % of budget in the mobility session, less than 2.5 % in the financial session, less than 10.5 % in the medical session, and practically none in the streaming session. Depending on the given analysis session UniTraX is able to save up to nearly all the budget that PINQ would have used. For clarity the x-axis does not start at zero, the y-axis does not end at one, and gridlines are removed.

global budget. As such it is an effort in the direction of finer grained budgets without the flexibility that UniTraX provides.

Figure 5.2 shows UniTraX's budget consumption relative to PINQ with partitioning. The difference between the two systems' budget consumptions has shrunk significantly. By giving up on a single global budget and using separate budgets on distinct partitions, PINQ is able to save large parts of records' privacy budgets.

UniTraX's additional savings depend on the specific queries asked in each analysis session. In the mobility session half of the queries concern taxi rides only in Manhattan, in the financial session queries mainly investigate female clients, and in the medical session the major focus is on few severely ill patients. These circumstances determine the amount of budget UniTraX saves. These savings are reflected in the shapes of the respective curves.

### 5.2.7. Absolute and relative latency overheads

In the following paragraphs I present end-to-end latencies for the three experimental setups: Direct, PINQ, and UniTraX. For each analysis session I show absolute latencies for each setup as well as UniTraX's relative latency overheads in comparison to the other two systems. End-to-end means the time between an analyst posing a query and receiving the corresponding answer.

All presented graphs are generated from the data of eight repeated experiment executions. I throw away the first execution of each experiment, as it is used to warm caches and thus prevent measurement noise from startup overheads. For each measured value of each query I collect the values of the other seven executions, sort them, throw away the smallest and the largest value, and obtain the final result by calculating the average out of the remaining five values. Each point in my graphs is thus an average, i.e., mean, of five values with the most extreme values removed beforehand. Any error bars indicate the standard deviation over the five runs.

**Mobility session.** Figure 5.3 presents absolute end-to-end latencies for a random 5 % sample of the 1213 queries of the mobility session. Queries are sorted on the x-axis by increasing latency with respect to the Direct system. Overheads are moderate. As expected, UniTraX is slower than PINQ, which is slower than Direct query execution without any privacy protection.

In Figure 5.4 I present the corresponding CDF for all 1213 queries in terms of the overhead of UniTraX relative to Direct and PINQ respectively. In half

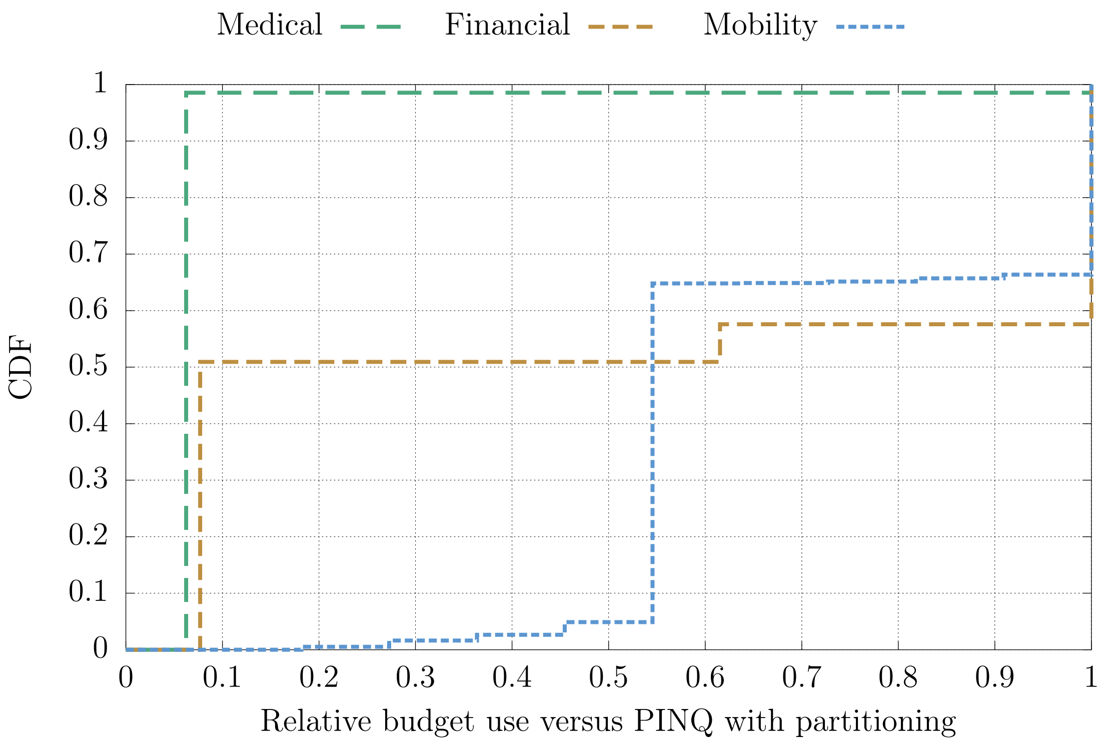


Figure 5.2.: CDF of relative budget use of UniTraX versus PINQ with partitioning. At the 50th-percentile UniTraX uses less than 55 % of budget in the mobility session, less than 8 % in the financial session, and less than 6.5 % in the medical session. There are no numbers for the streaming session as PINQ does not support dynamic datasets with the partitioning operation. In comparison to Figure 5.1 it is obvious that partitioning improves PINQ's budget efficiency by a large margin. However, for the three presented analysis sessions UniTraX is able to save more budget than PINQ.

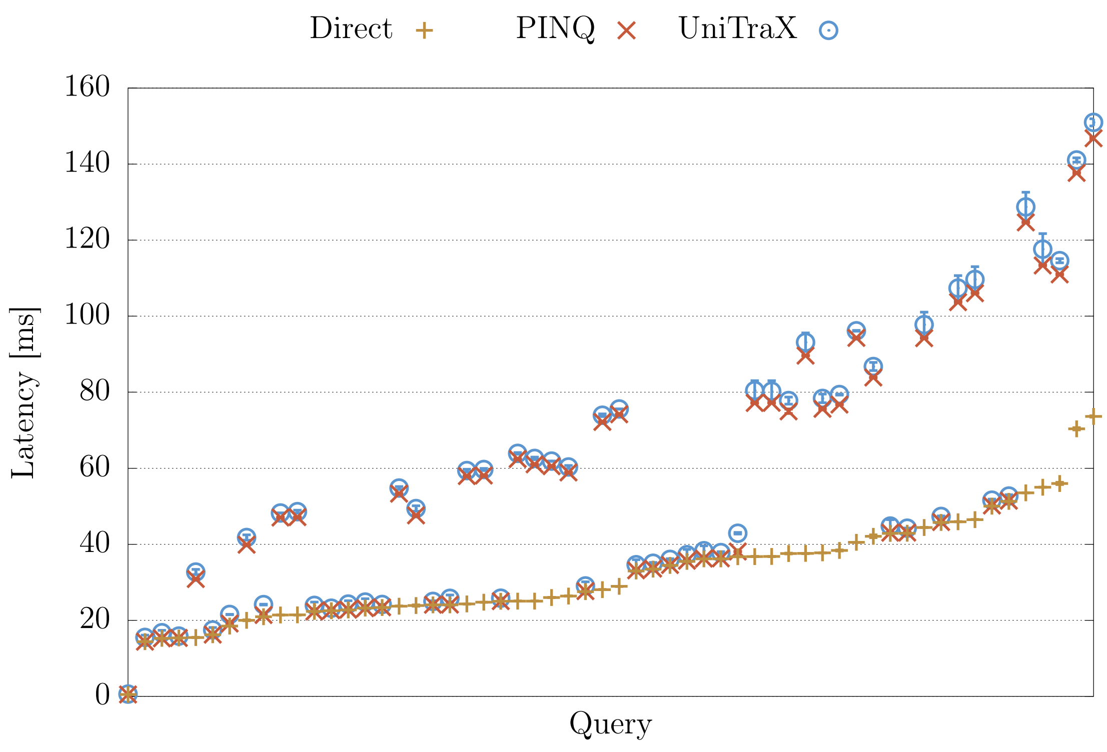


Figure 5.3.: End-to-end latencies of a 5 % sample of the 1213 queries in the mobility session. Queries are ordered according to latencies of the Direct system. Error bars indicate the standard deviation over five runs. The trend in the order of performance is evident. UniTraX is slower than PINQ, which is slower than Direct.

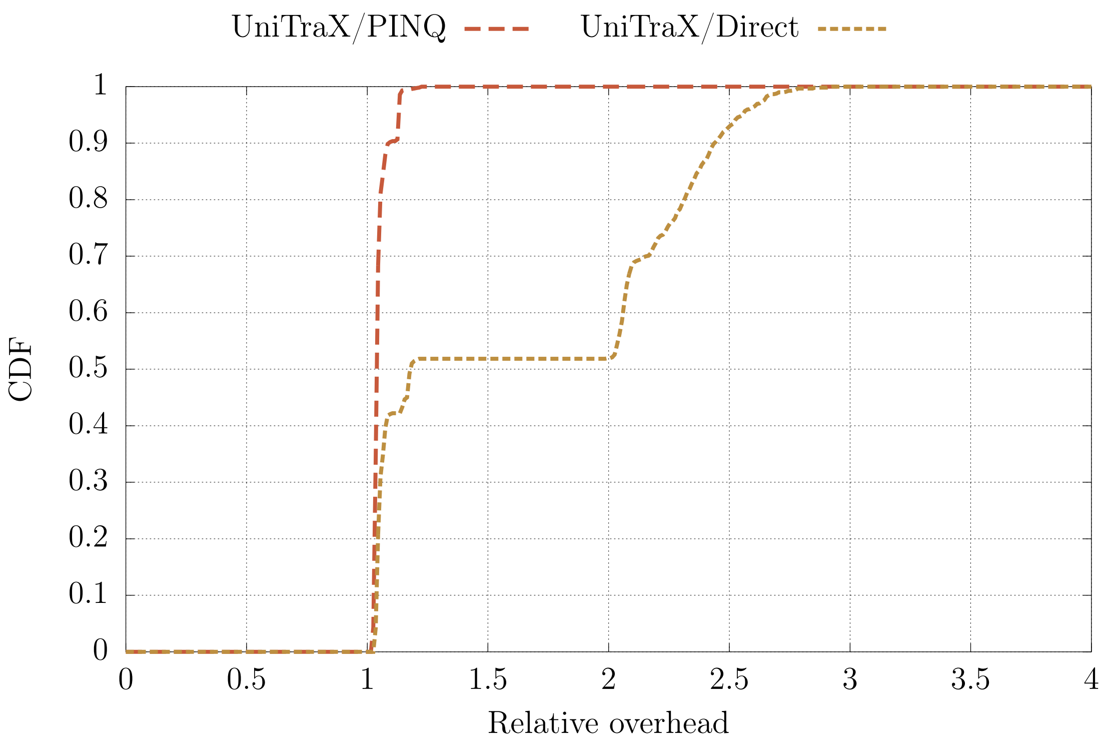


Figure 5.4.: CDF of relative end-to-end latency overheads incurred by UniTraX across all 1213 queries of the mobility session. At the 99th-percentile UniTraX is 1.13 x slower than PINQ and 2.72 x slower than the Direct system.

of the cases, UniTraX is 1.03 x slower than PINQ and 1.17 x slower than the Direct system. At the 99th-percentile UniTraX is 1.13 x slower than PINQ and 2.72 x slower than the Direct system. On average, UniTraX is 1.04 x slower than PINQ and 1.64 x slower than the Direct system. In summary, latency overheads introduced by UniTraX are moderate compared to a no privacy system and low compared to a global budget systems.

**Financial session.** Figure 5.5 shows absolute end-to-end latencies for a random 10 % sample of the 512 queries of the financial session. Queries are again sorted on the x-axis by increasing latency with respect to the Direct system. The numbers confirm the previously identified trend. However, due to the small measurements

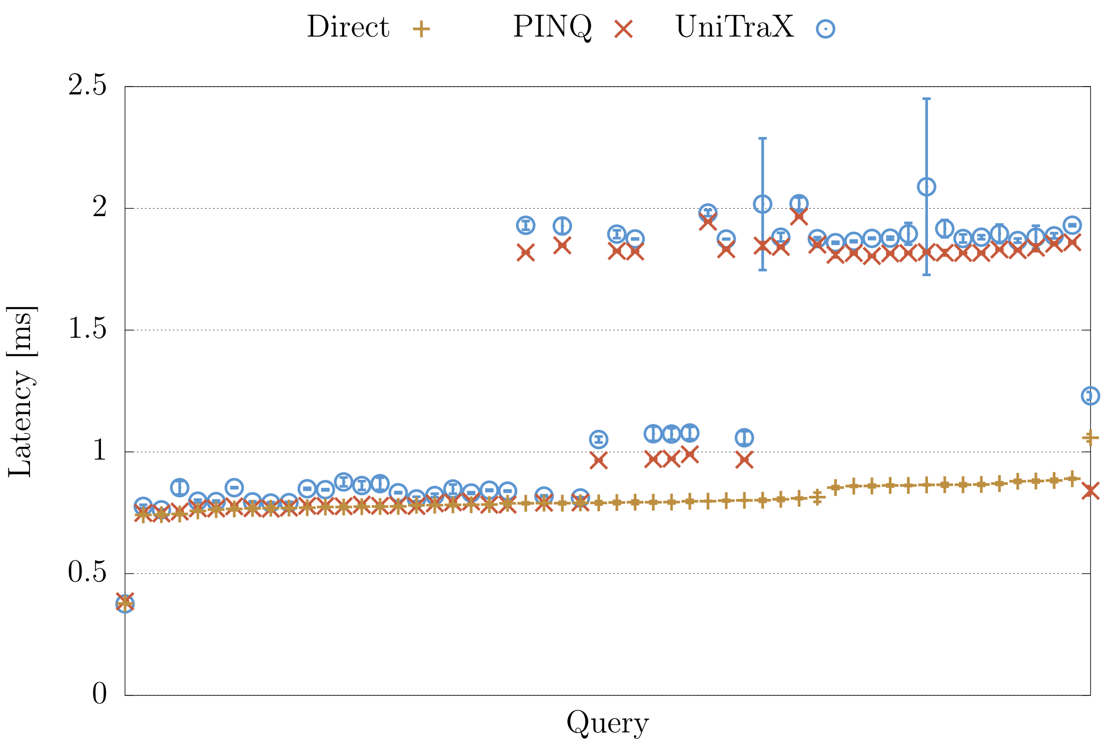


Figure 5.5.: End-to-end latencies of a 10 % sample of the 512 queries of the financial session. Queries are ordered according to latencies of the Direct system. Error bars indicate the standard deviation over five runs.

naturally the noise in the measurements is larger relative to the measured values. Error bars indicate the standard deviation over five runs.

Figure 5.6 shows the corresponding CDF for all 512 queries in terms of the relative overheads. At the median, UniTraX is 1.04 x slower than PINQ and 1.34 x slower than the Direct system. At the 99th-percentile UniTraX is 1.15 x slower than PINQ and 2.52 x slower than the Direct system. On average, UniTraX is 1.05 x slower than PINQ and 1.67 x slower than the Direct system. These numbers confirm the findings in the mobility session.

**Medical session.** Figure 5.7 presents absolute end-to-end latencies for a random 30 % sample of the 157 queries of the medical session. Queries are sorted by increasing latency with respect to the Direct system.

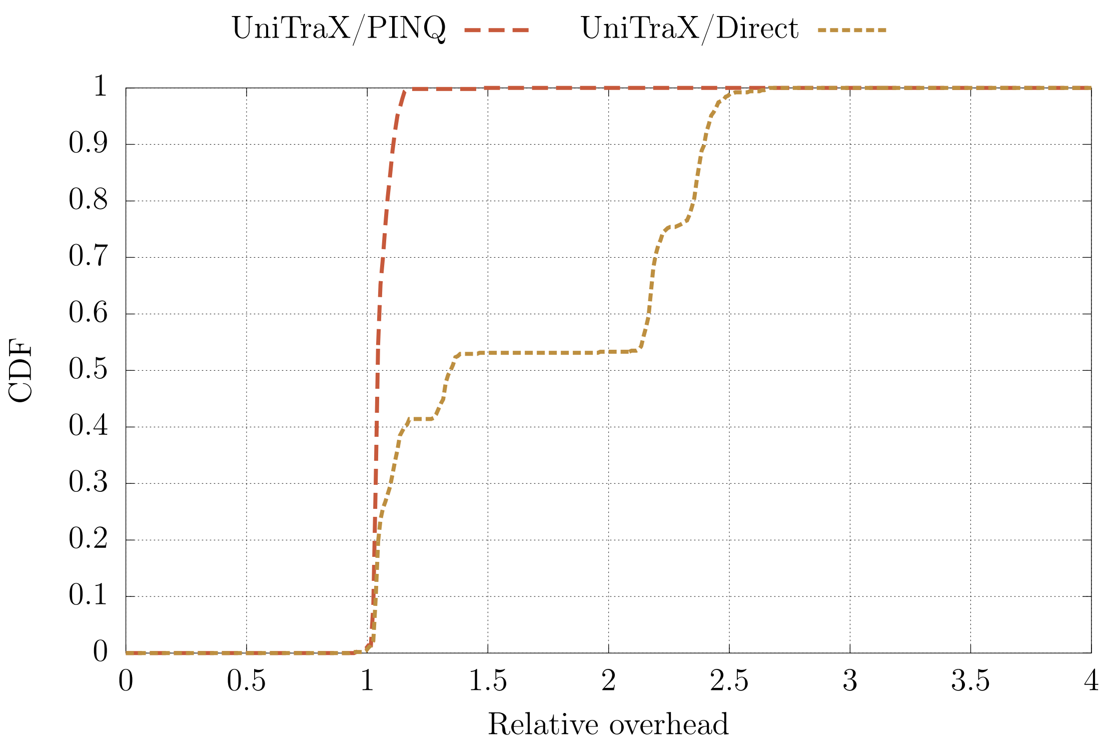


Figure 5.6.: CDF of relative end-to-end latency overheads incurred by UniTraX across all 512 queries of the financial session. At the 99th-percentile UniTraX is 1.15 x slower than PINQ and 2.52 x slower than the Direct system.

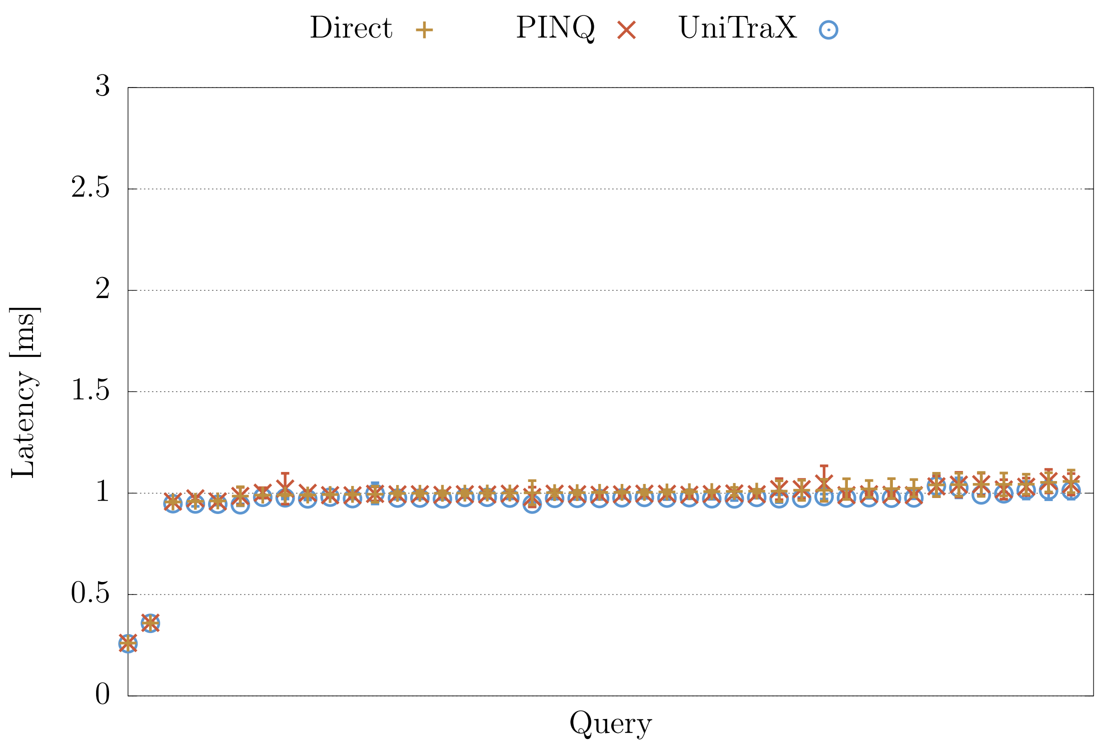


Figure 5.7.: End-to-end latencies of a 30 % sample of the 157 queries of the medical session. Queries are ordered according to latencies of the Direct system. Error bars indicate the standard deviation over five runs.

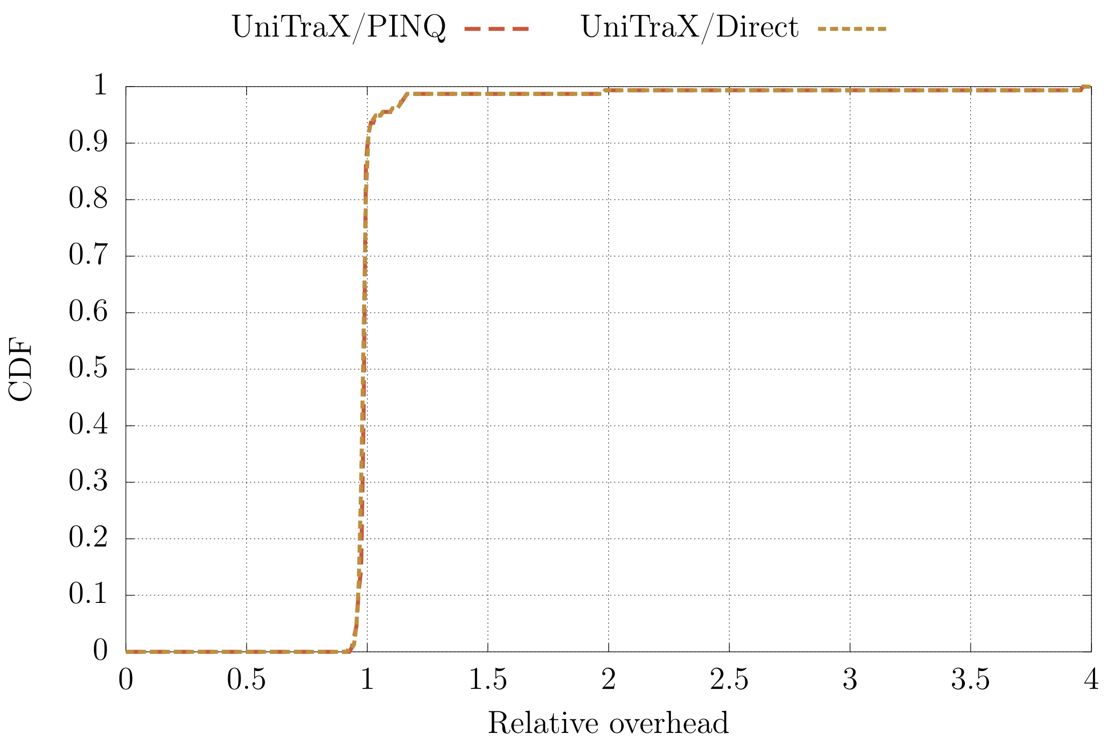


Figure 5.8.: CDF of relative end-to-end latency overheads incurred by UniTraX across all 157 queries of the medical session. At the 99th-percentile UniTraX is 2.80 x slower than PINQ and 2.81 x slower than the Direct system. However, these numbers must be viewed with caution as the absolute numbers are small (see Figure 5.7).

Figure 5.8 contains the corresponding CDF of the relative overheads for the 157 queries. At the median, UniTraX is 0.98 x slower than both PINQ and the Direct system, i.e., it is actually a bit faster. At the 99th-percentile UniTraX is 2.80 x slower than PINQ and 2.81 x slower than the Direct system. On average, UniTraX is 1.01 x slower than PINQ and the Direct system. Provided that the absolute numbers are small (see Figure 5.7) and that queries are executed on a remote database system, the observed minor speedup of UniTraX in comparison to the baseline systems must be viewed as within measurement variability.

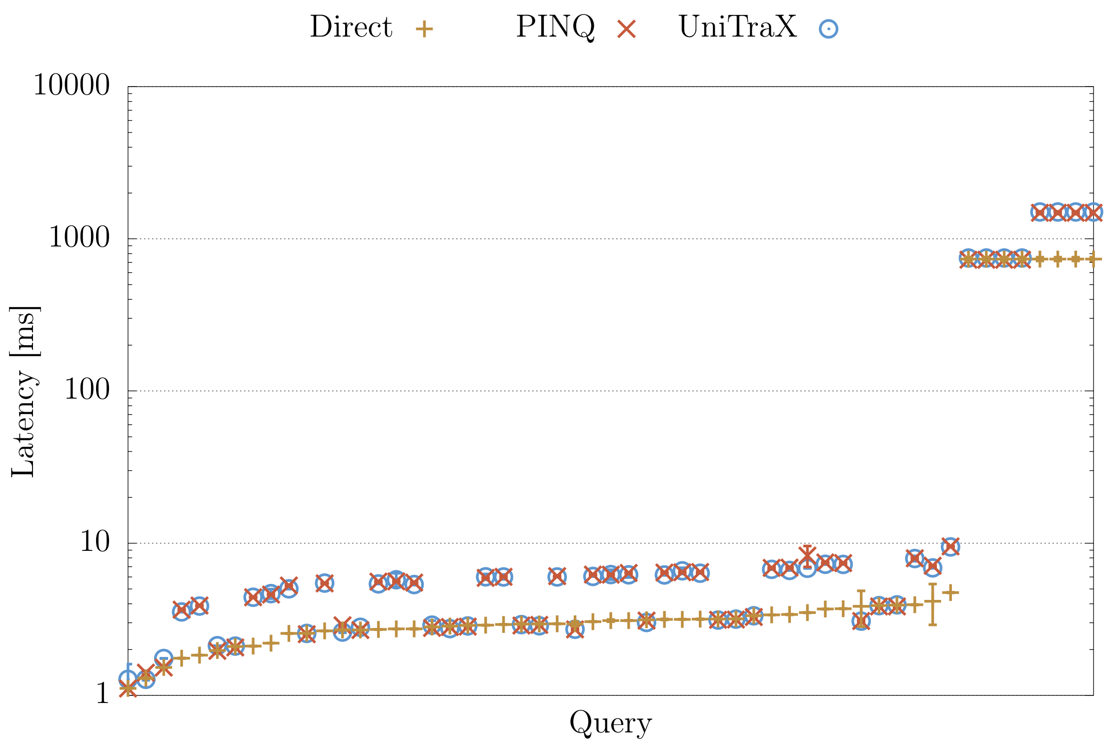


Figure 5.9.: End-to-end latencies of a 2 % sample of the 3067 queries of the streaming session. Queries are ordered according to latencies of the Direct system. Error bars indicate the standard deviation over five runs. The y-axis is presented in log scale as execution times of the monthly analysis are significantly higher than those of the hourly analyses.

**Streaming session.** Figure 5.9 shows absolute end-to-end latencies for a random 2 % sample of the 3067 queries of the streaming session. Queries are sorted by increasing latency with respect to the Direct system. The y-axis is presented in log scale as execution times of the monthly analysis are significantly higher than those of the hourly analyses.

Figure 5.10 shows the corresponding CDF of the relative overheads for the 3067 queries of the streaming session. At the median, UniTraX is as fast as PINQ and 1.97 x slower than the Direct system. At the 99th-percentile UniTraX is 1.15 x slower than PINQ and 2.17 x slower than the Direct system. On average, UniTraX is as fast as PINQ and 1.56 x slower than the Direct system.

The graph in Figure 5.10 further shows a speedup of UniTraX with respect to both Direct and PINQ. Both become significantly slower during their own specific short period of time. These periods do not overlap and consistently occur at their own specific time during the analysis session. Queries and results are identical to the other, non-affected systems during these periods. At this point I can only speculate as to why this happens.

**Summary.** Across the different analysis sessions UniTraX shows average overheads of less than 70 % compared to a no-privacy baseline and up to 5 % compared to a global budget system.

### 5.2.8. Latency overheads for PINQ partitioning

PINQ's partitioning operation always materializes and caches the resulting subsets of records immediately. To provide a fair comparison I additionally ran all my experiments in a version where all systems employ identical local caching. In the interest of space I solely present results for the mobility session.

Figure 5.11 shows absolute end-to-end latencies for a 5 % sample of the 1213 queries of the mobility session. Queries are sorted by increasing latency with respect to the Direct system. All systems cache the data of different partitions locally. For most queries the database server is not involved. Performance enhancing features of the server, e.g., indexes, are thus not available. However, there are no network overheads either.

In comparison to Figure 5.3 one can observe the same trends. However, absolute query times have significantly decreased due to the use of caching. Then again, relative distances between the different systems have increased. These increases are due to less efficient local query execution as well as the omission of network and database overheads.

Figure 5.12 shows the corresponding CDF of the relative overheads for the 1213 queries of the mobility session when PINQ uses the partitioning operation and all systems run on locally cached data. The x-axis is rendered in log scale to accommodate the increased slow-downs compared to the Direct system. At the median, UniTraX is 1.67 x slower than PINQ and 5.80 x slower than the Direct system. At the 99th-percentile UniTraX is 4.53 x slower than PINQ and 111 x slower than the Direct system. On average, UniTraX is 2.11 x slower than PINQ and 9.07 x slower than the Direct system.

The CDF confirms my previous finding. The trends stay the same but the distances between the different systems increase. The large slow-down compared

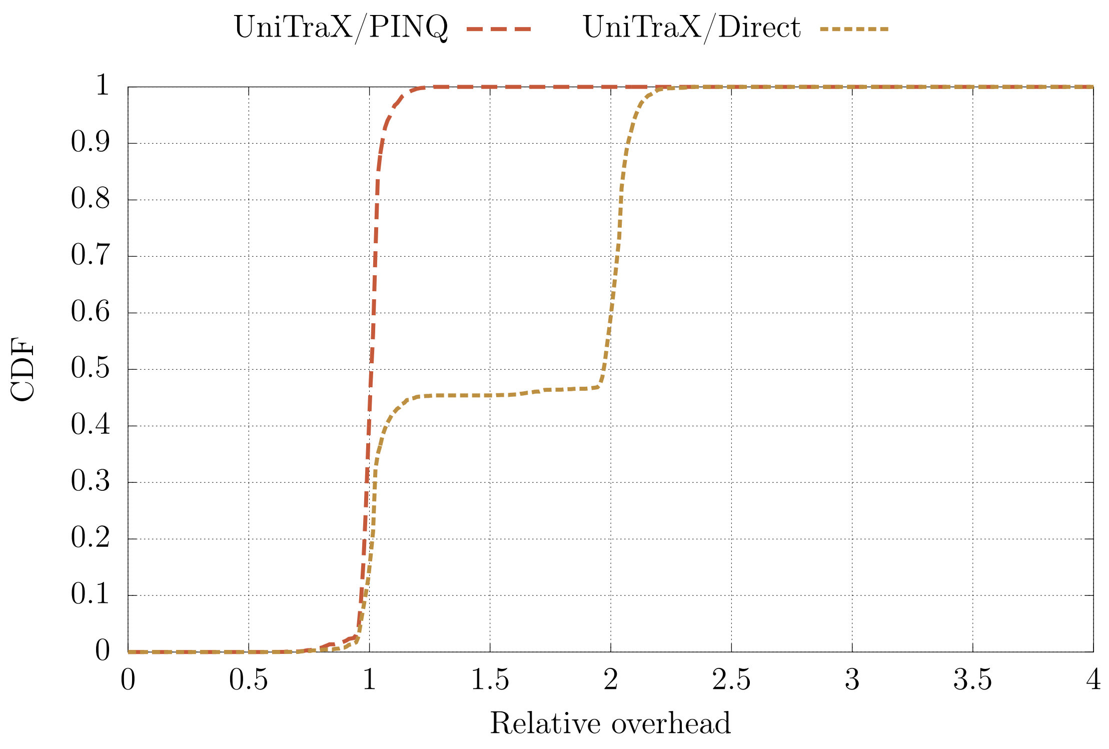


Figure 5.10.: CDF of relative end-to-end latency overheads incurred by UniTraX across all 3067 queries of the streaming session. At the 99th-percentile UniTraX is 1.15 x slower than PINQ and 2.17 x slower than the Direct system. The observable speedup is caused by the Direct system and PINQ becoming significantly slower during a short period of time.

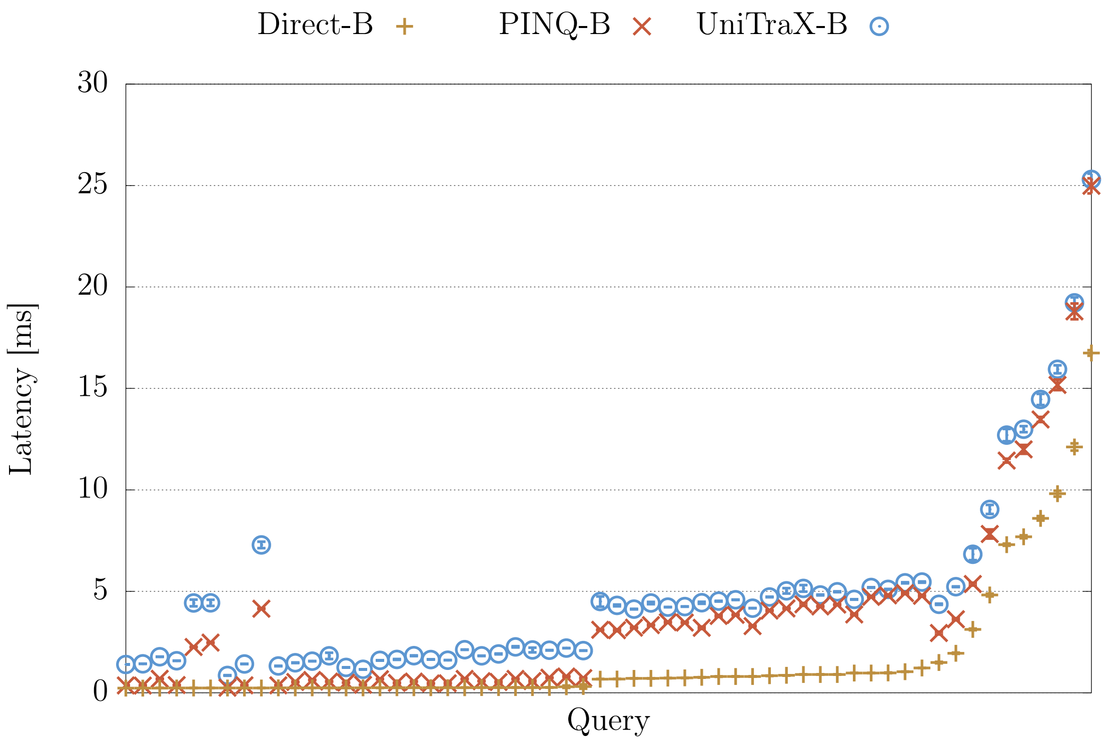


Figure 5.11.: End-to-end latencies of a 5 % sample of the 1213 queries of the mobility session. Queries are ordered according to latencies of the Direct system. Error bars indicate the standard deviation over five runs. PINQ employs its partitioning operation to safe further budgets. All systems cache the data of different partitions locally. Although absolute numbers are lower than with the database server, differences between the different systems have increased (compare Figure 5.3).

to the Direct system is due to the Direct system benefitting more from local caching. UniTraX performs worse than PINQ because of its additional budget check. I suspect that without the help of an index the local query executor checks all records sequentially.

The curve showing UniTraX's performance relative to PINQ's is slightly above zero at 1 with a tail of inverse slow down before. These speed-ups are caused by the eight partitioning queries in the mobility session. During these queries the database server performs the actual partitioning of the data and sends the individual partitions of data to the client for local caching. Overheads of partitioning are significantly larger for PINQ due to its use of a partitioning function in a group by operation. Such partitioning function must be evaluated on every record, which results in a full table scan. UniTraX and the Direct system on the other hand are able to leverage indexes as their partitioning is based on ranges.

### 5.2.9. Total query times and clean-up mechanisms

Figure 5.13 shows total query times. These measure from one query's start to the next query's start and include not only afore reported query latencies but also each query's respective clean-up time. The presented 5 % sample of the 1213 queries of the mobility session are the same as in Figure 5.4, but in execution order. Queries are thus not ordered according to the Direct system's end-to-end latencies, but shown in the order in which they execute during the mobility session.

Although the order is different, UniTraX's additional overheads in comparison to Figure 5.4 are evident. These are for most parts due to the clean-up overheads of the internal history structure. The larger the structure becomes over time, the higher the resulting clean-up overhead. However, without any cleaning UniTraX quickly experiences query latency overheads. In that case analysts must wait longer for their results, which is not desirable.

Figure 5.13 further presents the effects of the two different clean-up mechanisms, cleaning threshold and re-balancing (see "Clean-up mechanisms" at the end of section 5.1). The system labelled "UniTraX" represents the default version that executes a clean-up of internal history data structures after every query. "U-Threshold", in contrast, waits until 500 additional subspaces have accumulated before a clean-up is performed. Note that this does not necessarily mean that 500 additional queries must execute, as a single query can—and regularly does—result in many additional subspaces. "U-Re-balancing" uses additional budget to allow for further compression of the internal history data structures. Nevertheless, it performs a clean-up after every query. Finally, "U-both" combines both

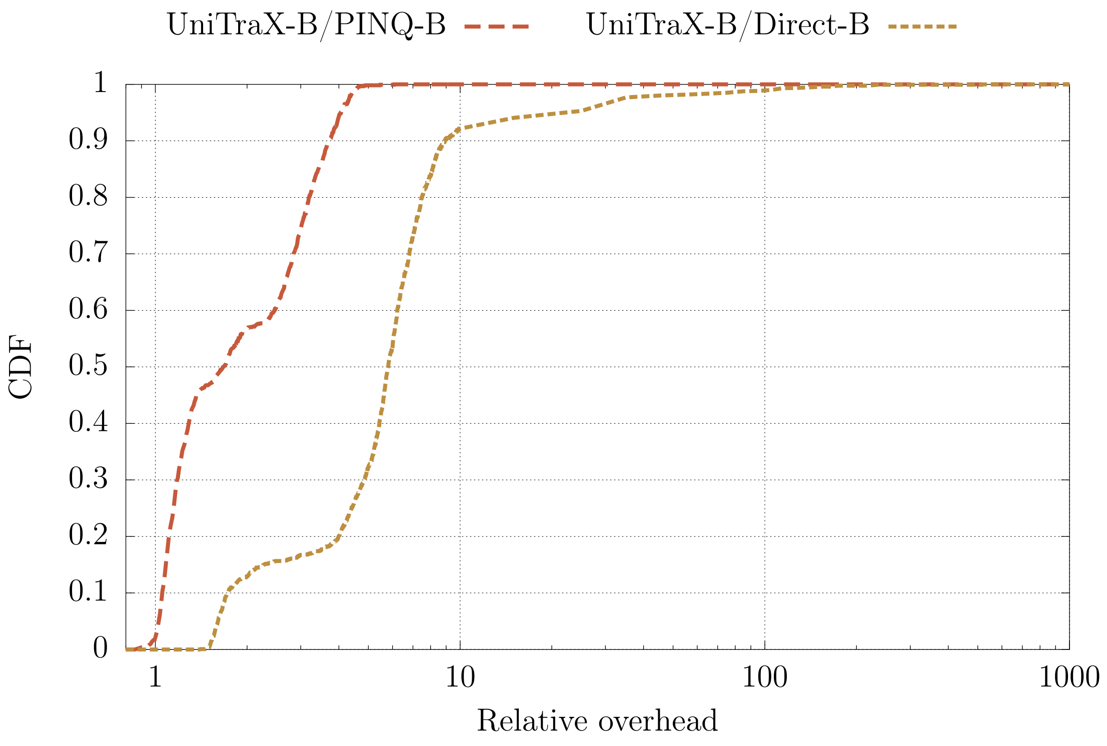


Figure 5.12.: CDF of relative end-to-end latency overheads incurred by UniTraX across all 1213 queries of the mobility session. PINQ is configured to use partitioning and all systems cache data locally. Note that the x-axis starts at 0.8 and is presented in log scale. At the 99th-percentile UniTraX is 4.53 x slower than PINQ and 111 x slower than the Direct system. The curve that shows UniTraX relative to PINQ is slightly above zero at 1 due to the eight partitioning queries that take longer for PINQ.

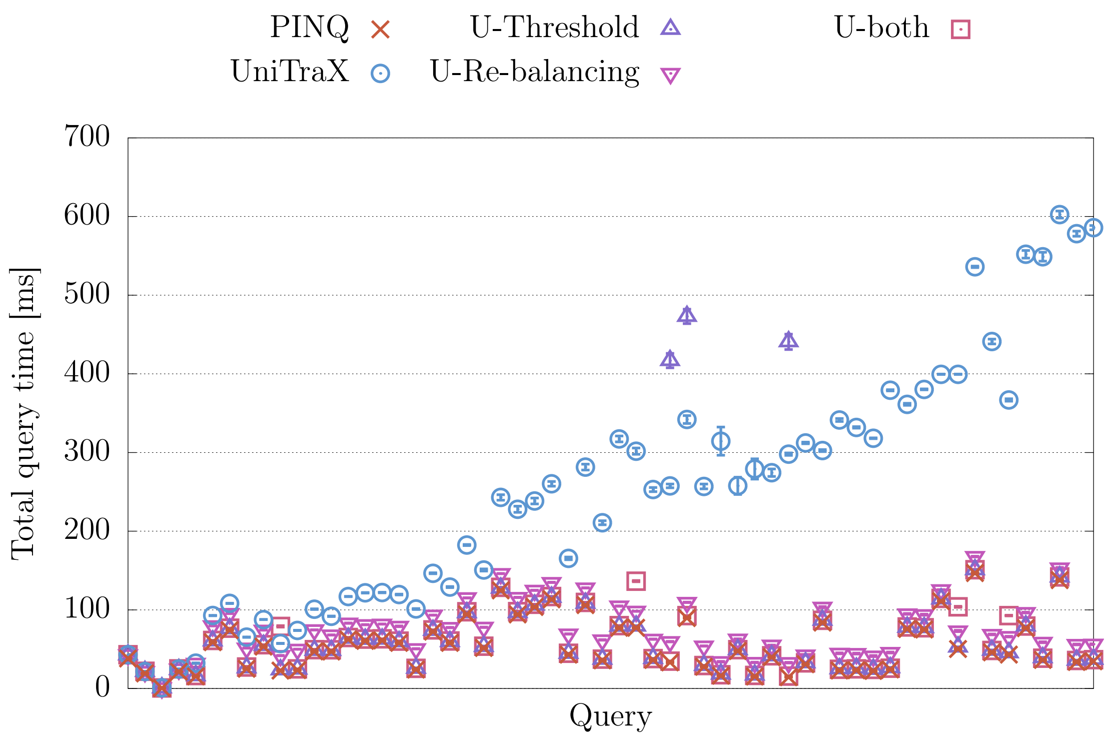


Figure 5.13.: Total query times including clean-up of a 5 % sample of the 1213 queries of the mobility session. Queries are in execution order. Error bars indicate the standard deviation over five runs. In comparison to Figure 5.4 the large clean-up overheads of UniTraX are evident. UniTraX with cleaning threshold performs at query execution latency until the threshold is reached. At that point a clean-up takes longer as additional work has accumulated. Re-balancing helps keeping UniTraX's internal state small and regular clean-ups fast. Nevertheless, the best performance is achieved by their combination.

techniques and uses re-balancing but waits for 500 additional subspaces before cleaning.

As expected, U-Threshold executes queries as fast as PINQ and thus faster than UniTraX, which gets cleaned after every query. However, when the threshold of additional subspaces is reached, the following clean-up takes significantly longer than UniTraX because of the accumulated additional subspaces.

U-Re-balancing regularly performs worse than U-Threshold. This behavior is expected as U-Re-balancing performs a clean-up after every query. Nevertheless, it does not suffer the occasional high overheads of U-Threshold when the threshold is reached and a clean-up has to be performed.

Despite a clean-up after every query, U-Re-balancing performs significantly better than UniTraX, due to its smaller internal state. The lowest total query times, however, are reached by U-both, as it combines both threshold and re-balancing.

Figure 5.14 shows the corresponding CDF of relative total-query-time overheads incurred by UniTraX across all 1213 queries of the mobility session. It confirms my finding that a combination of cleaning threshold and re-balancing performs closest to optimal. Both methods individually are better than pure UniTraX with clean-up after every query. However, both have their individual shortcomings. U-Threshold significantly diverts from the optimum for a small number of queries, i.e., the ones when a clean-up happens. U-Re-balancing outperforms U-Threshold on those queries but is slower for most of the remaining queries. The combination, U-both, is able to combine the advantages of both and stays close to the relative query latency overheads reported in Figure 5.4, i.e., close to the relative overhead without clean-up.

**Size of budget tracking state.** Figure 5.15 shows the number of subspaces tracked by UniTraX at the end of each query. Queries are again in execution order (compare Figure 5.13). These numbers do not change across different runs. The four curves represent the four possible combinations of the two clean-up mechanisms, cleaning threshold and re-balancing. They illustrate that the analyst can dramatically affect the size of the budget tracking state based on how queries are formulated.

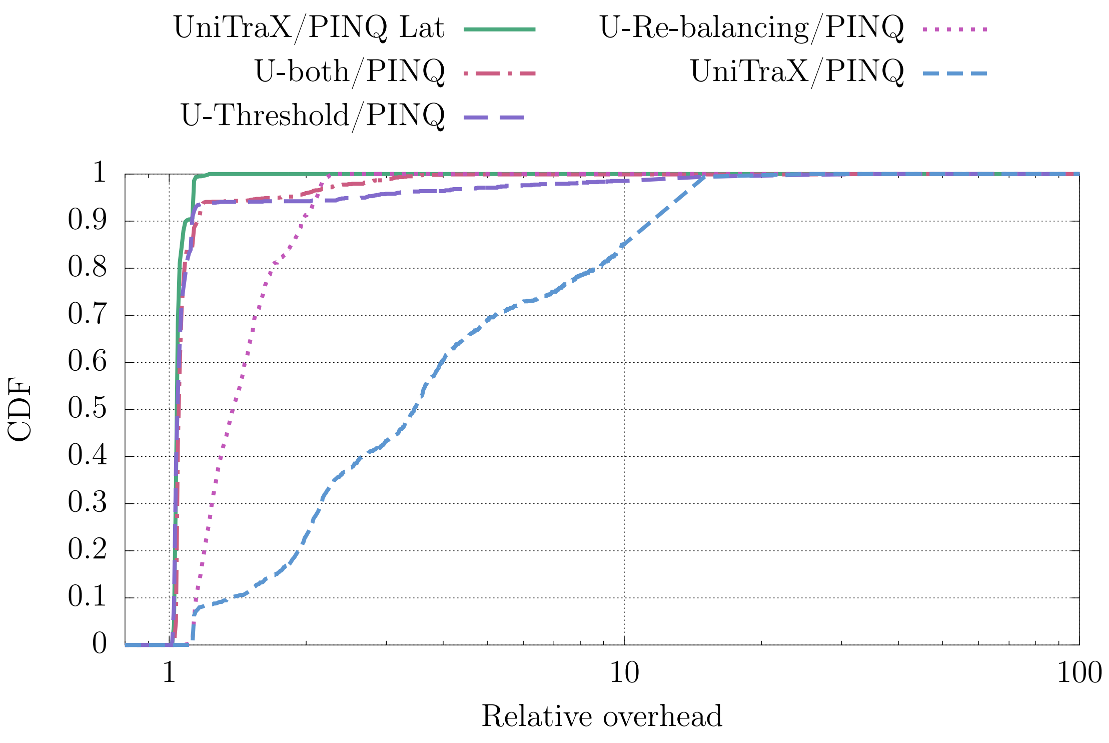


Figure 5.14.: CDF of relative total-query-time overheads incurred by UniTraX across all 1213 queries of the mobility session. At the 99th-percentile UniTraX is 19.39 x slower than PINQ. These overheads are largely due to the expensive clean-up operations performed by UniTraX once a query's result has been returned. Cleaning threshold (U-Threshold), re-balancing (U-Re-balancing), and their combination (U-both) help reducing UniTraX's total query time overheads, getting close to relative query latency overheads (UniTraX/PINQ Lat; from Figure 5.4), i.e., close to the relative overhead without clean-up.

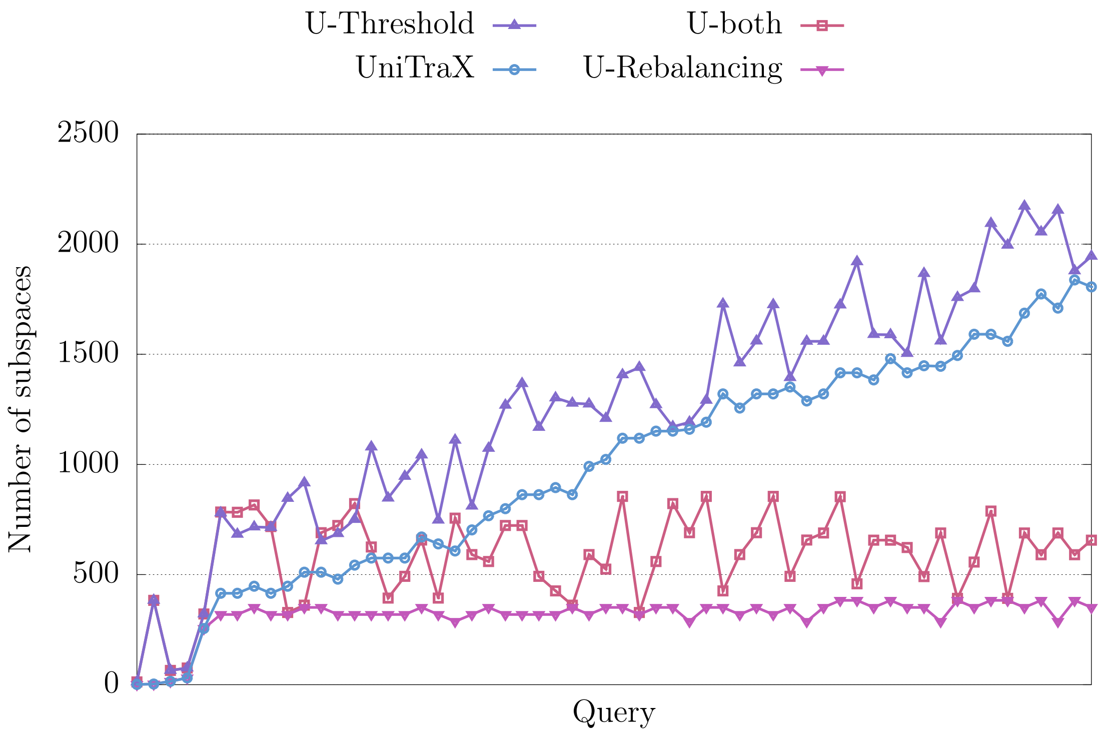


Figure 5.15.: Number of subspaces tracked by UniTraX for a 5 % sample of the 1213 queries of the mobility session. Queries are in execution order. Subspaces are counted after any clean-up, right before the next query. UniTraX accumulates more subspaces as queries execute and despite its constant cleaning efforts. As expected, U-Threshold collects more subspaces except for when the threshold is reached and a cleanup performed. U-Re-balancing, i.e., the version that uses budgets more evenly, is able to keep the number of subspaces relatively constant. Finally, U-both, the combination of cleaning threshold and re-balancing, allows for more fluctuation but returns back to the constant low number of subspaces every time the threshold is reached. These observations directly connect to the measured total query times in Figure 5.13.

## 5.3. Conclusion

This chapter presented a prototype implementation of UniTraX and its evaluation. It showed that UniTraX is able to save significant amounts of privacy budgets compared to global budget systems. The prototype implementation works on a range of different domains and incurs moderate overheads on realistic workloads.

There are several directions for future work. First, PINQ allows for arbitrary partitioning functions and additional work is needed to understand their full budget saving potential. 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.

Engineering-wise, the prototype implementation could be further enhanced in the following ways.

1. The prototype is sequential at the moment. Nevertheless, many functions could be parallelized. For instance, any operation on the internal history structure could work on different subspaces in parallel.

2. The history structure uses simple lists to store subspaces. However, optimized data structures for spatial data exist [10–12]. Such data structures could offer performance trade-offs that are currently not possible.

3. UniTraX does not support projection. The schema of the database needs to prevail, which is not guaranteed after general projection. However, UniTraX should be able to support a limited version, e.g., one that allows to project onto a subset of data fields.

4. Based on PINQ's use, UniTraX employs double and float data types for all numerical columns. However, practical experience with the system showed that inaccuracies in floating point arithmetic are a constant source of bugs and problems. Provided that UniTraX's tracking mechanism is discrete in nature, these could be changed to long and bigint datatypes, depending code in UniTraX be simplified, and values only be transformed back to double when used with PINQ.

# 6. Conclusion

Private analytics systems today come in two flavors, the preferred and the actually used. The preferred systems are based on the theory of DP and are able to provide *formal* guarantees of privacy. The actually used systems are based on *ad-hoc* techniques that often fail to adequately protect privacy. The latter thrive throughout industry as they fulfill analytic requirements and thus provide increased usability.

## 6.1. This thesis

In this thesis I have investigated usable private data analytics systems in the context of DP. I tackled two challenges to improve DP's usability. (1) Weakening DP's guarantees to increase result accuracy but maintain proper privacy protection. (2) Limiting privacy budget use to part of a dataset without sacrificing result accuracy. To this end, I used two different approaches to overcome these challenges. (1) I did away with the requirements of DP, started from the point of view of an attacker, and created a system design that allows higher result accuracy than DP but is hard to attack in practice. (2) I used my insights to improve DP systems and introduced UniTraX, the first DP system that allows to use privacy budgets on only parts of a dataset without sacrificing result accuracy.

**Leaving Differential Privacy.** My first approach consisted of two parts. First I created a system design with guaranteed utility in form of unlimited queries and fixed levels of noise added to results. Unfortunately, this design could still be easily attacked. However, using my insights I created a second design, which is close to DP but only provides a weakened version thereof. Despite the weakening I showed together with my adviser that attacking the system is hard in practice. One component of the design was QBB, a protective mechanism based only on queries.

**Returning to Differential Privacy.** In my second approach I built on the insights of the first. In particular, I extended and developed QBB into its DP successor—UniTraX. Together with my collaborators I developed a formal model and we presented a proof that UniTraX fulfills the rules of DP. UniTraX is the first DP system that allows to restrict usage of privacy budgets to parts of a dataset without giving up on result accuracy. In contrast to other systems, it is able to inform analysts about potential biases introduced by privacy protection itself. I implemented and evaluated a prototype and showed that UniTraX is able to save significant amounts of privacy budgets at average overheads below 70 % for realistic workloads across different domains.

## 6.2. The bigger picture

Taking a step back, the general problem at hand is the gap between DP worst case assumptions on one hand and the context and behavior of real data analytics tasks on the other. Previous DP systems typically assume that attackers know all records except the victim's. They assume that all queries against the system target the victim, and that all records are potential victims. Obviously, for real data analytics tasks these assumptions are too negative. Nevertheless, protective systems use these assumptions to dimension the safeguards, e.g. artificial noise added to results. It is not surprising that exaggerated assumptions and safeguards lead to diminished usability.

This thesis is a first step in improving DP systems' usability as it is a first step in closing the gap between worst case assumptions and real analytic tasks. UniTraX allows to limit the worst case assumptions about a query to only part of a dataset. Thus, records in the other parts are not considered victims with regards to that query. Therefore, their privacy budget does not need to be reduced and is available for further queries. By adjusting the worst case assumptions by a small bit, UniTraX is able to save significant amounts of privacy budgets and thus allow additional queries.

## 6.3. Future work

Following the insights of this thesis I suggest the following three directions of future work.

1. Derive a simple meaning for $\varepsilon$, starting from an attacker's perspective.

2. Safe additional budget by refining further worst case assumptions.
3. Extend and formally prove the idea of reducing artificial noise depending on an attacker's uncertainty about the data around a victim's record.

**Deriving a meaning for $\varepsilon$.** Despite different efforts [6, 63, 71, 72, 78], I am to date not aware of any simple meaning of $\varepsilon$. To find such a meaning, I suggest to continue from the findings of the first part of this thesis. There, I showed that even if DP's requirements are not always fulfilled it is hard to attack a system. I showed the huge amount of additional knowledge attackers need in order to only have a small chance of being successful. Attackers' chances and their confidence in having obtained private information are good starting points in understanding what specific system configurations—including $\varepsilon$—mean.

**Saving additional budgets.** For saving $\varepsilon$, i.e., saving privacy budgets, this thesis presented a first step. It showed how to make DP systems more usable by refining their worst case assumptions. However, DP makes and upholds many worst case assumptions. Most of them help simplify the problem of ensuring and proofing DP's guarantees. Nevertheless, I believe it to be possible to identify more cases where assumptions can be better fit to actual analyses at the benefit of increased budget savings.

**Replacing noise with uncertainty.** In the first part of this thesis I assumed attacker's to be uncertain about the data. I used the uncertainty about the data around a victim's record to reduce the artificial noise on the answers. Despite specific restrictions, e.g., queries were limited to counts, I believe the results to be promising. Building on these results, I suggest to extend and formalize the mechanism with the goal of proofing it formally correct. In the end, one would like to create a system that understands which uncertainty exists in the data and how it is structured. Such system could apply this knowledge to accurately reduce artificial noise and safe additional privacy budgets.

# A. Proofs of the formal results on UniTraX

We first show a lemma that will prove helpful in the proofs of our main results.

**Lemma 1** (Distributivity of Distance)**.** *For all sets $E, E', F, F' \in 2^{\mathcal{R}}$ the following statements hold:*

1. $(E \Delta E') \cap F = (E \cap F) \Delta (E' \cap F)$
2. $(E \cup F) \setminus (E' \cup F') \subseteq (E \setminus E') \cup (F \setminus F')$
3. $(E \cup F) \Delta (E' \cup F') \subseteq (E \Delta E') \cup (F \Delta F')$
4. $(E \setminus F) \setminus (E' \setminus F') \subseteq (E \setminus E') \cup (F' \setminus F)$
5. $(E \setminus F) \Delta (E' \setminus F') \subseteq (E \Delta E') \cup (F \Delta F')$.

*Proof.* The proof uses simple set theory operations.

1. We show statement 1 as follows:

$$\begin{aligned}
(E \Delta E') \cap F &= ((E \setminus E') \cup (E' \setminus E)) \cap F \\
&= ((E \setminus E') \cap F) \cup ((E' \setminus E) \cap F) \\
&= (((E \setminus E') \cap F) \cup \emptyset) \cup (((E' \setminus E) \cap F) \cup \emptyset) \\
&= (((E \setminus E') \cap F) \cup (E \cap \emptyset)) \cup (((E' \setminus E) \cap F) \cup (E' \cap \emptyset)) \\
&= (((E \setminus E') \cap F) \cup (E \cap (F \setminus F))) \cup \\
&\quad\ (((E' \setminus E) \cap F) \cup (E' \cap (F \setminus F))) \\
&= ((E \cap F) \setminus E') \cup ((E \cap F) \setminus F) \cup \\
&\quad\ ((E' \cap F) \setminus E) \cup ((E' \cap F) \setminus F) \\
&= ((E \cap F) \setminus (E' \cap F)) \cup ((E' \cap F) \setminus (E \cap F)) \\
&= (E \cap F) \Delta (E' \cap F)
\end{aligned}$$

2. We show statement 2 as follows:

$$
\begin{aligned}
(E \cup F) \setminus (E' \cup F') &= (E \cup F) \cap (E' \cup F')^{-1} \\
&= (E \cup F) \cap (E'^{-1} \cap F'^{-1}) \\
&= (E \cap (E'^{-1} \cap F'^{-1})) \cup (F \cap (E'^{-1} \cap F'^{-1})) \\
&= ((E \setminus E') \cap F'^{-1}) \cup ((F \setminus F') \cap E'^{-1}) \\
&\subseteq (E \setminus E') \cup (F \setminus F')
\end{aligned}
$$

3. Statement 3 follows immediately from the defintion of $\Delta$ by applying statement 2 twice.

4. We show statement 4 as follows:

$$
\begin{aligned}
(E \setminus F) \setminus (E' \setminus F') &= (E \cap F^{-1}) \cap (E' \cap F'^{-1})^{-1} \\
&= (E \cap F^{-1}) \cap (E'^{-1} \cup (F'^{-1})^{-1}) \\
&= (E \cap F^{-1}) \cap (E'^{-1} \cup F') \\
&= (E \cap F^{-1} \cap E'^{-1}) \cup (E \cap F^{-1} \cap F') \\
&= ((E \setminus E') \cap F'^{-1}) \cup ((F' \setminus F) \cap E) \\
&\subseteq (E \setminus E') \cup (F' \setminus F)
\end{aligned}
$$

5. Statement 5 follows immediately from the defintion of $\Delta$ by applying statement 4 twice.

□

Furthermore, we note the following observation that follows immediately from the definition of the history in the semantic rules (Query) and (Query-Drop). This will prove useful in the proof of the main results.

**Observation A.0.1** (History Continuously Increasing)**.** *For all configurations* $\mathbb{C} = (P, E, H, T), \mathbb{C}' = (P', E', H', T') \in \mathsf{Config}$*, all traces* $\sigma \in \mathsf{Act}^*$*, and all probabilities* $p \in [0, 1]$ *such that* $\mathbb{C} \overset{\sigma}{\Longrightarrow}_p \mathbb{C}'$ *it holds that* $H(r) \leq H'(r)$ *for all* $r \in \mathcal{R}$*.*

In order to show the privacy of UniTraX (Theorem 1), we first show the following strong invariant of configurations that takes into account how UniTraX tracks the consumption history. We show two different theorems: one for a single configuration step and one that extends the invariant to traces.

**Theorem 2** (Configuration Invariant (Single Step)). *Let* $\mathbb{C}_0 = (P_0, E_0, H_0, T_0)$, $\mathbb{C}'_0 = (P'_0, E'_0, H'_0, T'_0)$, $\mathbb{C}_1 = (P_1, E_1, H_1, T_1)$, $\mathbb{C}'_1 = (P'_1, E'_1, H'_1, T'_1) \in \mathsf{Config}$ *be four configurations,* $\alpha_0, \alpha_1 \in \mathsf{Act}$ *two action labels,* $p_0, p_1 \in [0, 1]$ *two probabilities, and* $R_\alpha, R \in 2^{\mathcal{R}}$ *two sets of records.*

*If*

*(P.1) for* $i \in \{0, 1\}$ *it holds that* $\mathbb{C}_i \xrightarrow{\alpha_i}_{p_i} \mathbb{C}'_i$ *and*

*(P.2)* $P_0 = P_1$ *and* $H_0 = H_1$ *and* $T_0 = T_1$ *and*

*(P.3)* $\mathit{dist}(\alpha_0, \alpha_1) = R_\alpha$ *and*

*(P.4)* $E_0 \Delta E_1 = R$ *and*

*(P.5) for all* $r \in \mathcal{R}$ *it holds that* $H_0(r) \leq r.c_B$

*then*

*(C.1)* $P'_0 = P'_1$ *and* $H'_0 = H'_1$ *and* $T'_0 = T'_1$ *and*

*(C.2)* $E'_0 \Delta E'_1 \subseteq R_\alpha \cup R$ *and*

*(C.3)* $p_0 \leq p_1 \cdot e^{\sum_{r \in R_\alpha \cup R} H'_0(r) - H_0(r)}$ *and*

*(C.4) for all* $r \in \mathcal{R}$ *it holds that* $H'_0(r) \leq r.c_B$.

*Proof.* Proof by case distinction on label $\alpha_i$, i.e., the applied operational semantics rule. We note that since $\mathit{dist}(\alpha_0, \alpha_1)$ is defined, the applied rule in both configuration steps must have been the same. We rely on the fact that the analyst is by definition internally deterministic and deadlock-free.

**Case: Deterministic rules (Select), (Read-History), and (Reject).** We note that in these cases $\alpha_0 = \alpha_1$ and by definition of *dist* it holds that $R_\alpha = \mathit{dist}(\alpha_0, \alpha_1) = \emptyset$. By inspection of the rules and the fact that the analyst is by definition internally deterministic and deadlock-free, it immediately follows that $P'_0 = P'_1$, $H_0 = H'_0 = H'_1 = H_1$, and $T'_0 = T'_1$ (C.1). Furthermore, we note that $p_0 = p_1 = 1$ and since $H'_0 = H_0$ we can immediately conclude both (C.3) and (C.4) (by (P.5)). We also observe that in all these cases $E'_0 = E_0$ and $E'_1 = E_1$, i.e., $E'_0 \Delta E'_1 = E_0 \Delta E_1 = R$ and thus $E'_0 \Delta E'_1 \subseteq R_\alpha \cup R$ (C.2).

**Case: Deterministic rule (Update).** In this case we know that $P'_0 = P'_1$, $H_0 = H'_0 = H'_1 = H_1$, and $T_0 = T'_0 = T'_1 = T_1$ (C.1). Furthermore, we note that $p_0 = p_1 = 1$ and since $H'_0 = H_0$ we can immediately conclude both (C.3) and (C.4) (by (P.5)). Additionally, we know that $\alpha_i = R^i_{in} : R^i_{del}$ for some sets of records $R^i_{\mathsf{in}}, R^i_{\mathsf{del}} \in 2^{\mathcal{R}}$ and $i \in \{0, 1\}$. Thus, by definition of *dist* it must be the case that

$$R_\alpha = dist(\alpha_0, \alpha_1) = (R^0_{\mathsf{in}} \Delta R^1_{\mathsf{in}}) \cup (R^0_{\mathsf{del}} \Delta R^1_{\mathsf{del}}).$$

This also means that

$$E'_i = (E_i \cup R^i_{\mathsf{in}}) \setminus R^i_{\mathsf{del}}.$$

It remains to be shown that

$$E'_0 \Delta E'_1 \subseteq R_\alpha \cup R.$$

This is equivalent to showing that

$$\begin{aligned}&((E_0 \cup R^0_{\mathsf{in}}) \setminus R^0_{\mathsf{del}}) \Delta ((E_1 \cup R^1_{\mathsf{in}}) \setminus R^1_{\mathsf{del}}) \\ &\subseteq ((R^0_{\mathsf{in}} \Delta R^1_{\mathsf{in}}) \cup (R^0_{\mathsf{del}} \Delta R^1_{\mathsf{del}})) \cup (E_0 \Delta E_1).\end{aligned}$$

We apply statement 5 of Lemma 1 to deduce that

$$\begin{aligned}&((E_0 \cup R^0_{\mathsf{in}}) \setminus R^0_{\mathsf{del}}) \Delta ((E_1 \cup R^1_{\mathsf{in}}) \setminus R^1_{\mathsf{del}}) \\ &\subseteq ((E_0 \cup R^0_{\mathsf{in}}) \Delta (E_1 \cup R^1_{\mathsf{in}})) \cup (R^0_{\mathsf{del}} \Delta R^1_{\mathsf{del}}).\end{aligned}$$

We then apply statement 3 of Lemma 1 to deduce that

$$\begin{aligned}&((E_0 \cup R^0_{\mathsf{in}}) \Delta (E_1 \cup R^1_{\mathsf{in}})) \cup (R^0_{\mathsf{del}} \Delta R^1_{\mathsf{del}}) \\ &\subseteq (E_0 \Delta E_1) \cup (R^0_{\mathsf{in}} \Delta R^1_{\mathsf{in}}) \cup (R^0_{\mathsf{del}} \Delta R^1_{\mathsf{del}})\end{aligned}$$

and conclude.

**Case: Probabilistic query rule (Query).** By the definition of *dist* we know that there must exist some value $n \in \mathsf{Val}$ such that $\alpha_0 = \alpha_1 = n$ and $R_\alpha = dist(\alpha_0, \alpha_1) = \emptyset$. Using our previous knowledge and the definition of (QUERY) we can immediately observe that $P'_0 = P'_1$, $H'_0 = H'_1$, $T'_0 = T'_1$ and deduce (C.1). We also know that $E'_i = E_i$ for $i \in \{0, 1\}$ and thus $E'_0 \Delta E'_1 = E_0 \Delta E_1 \subseteq R \subseteq R_\alpha \cup R$ (C.2). We know that for all $r \in \mathcal{R}$ the history $H'_0 = H'_1$ is defined as $H'_0(r) \overset{\text{def}}{=} H_0(r)$ if $\neg sspace(r, H_0(r))$ and as $H'_0(r) \overset{\text{def}}{=} H_0(r) + \varepsilon$ if $sspace(r, H_0(r))$ is true. In the latter case of $sspace(r, H_0(r))$, the premises of (QUERY) additionally guarantee that $H_0(r) + \varepsilon \leq r.c_B$, which lets us immediately conclude that $H'_0(r) \leq r.c_B$, for all $r \in \mathcal{R}$ such that $sspace(r)$. Using (P.5) we can directly deduce that $H_0'(r) \leq r.c_B$, for all $r \in \mathcal{R}$ (C.4). The probabilities $p_i$ are defined as

$$\mathrm{Prob}[Q_\varepsilon(E_i|_{sspace,H_0}) = n].$$

By the differential privacy of the underlying query mechanism $Q_\varepsilon(sv)$ it must thus be the case that

$$p_0 \leq p_1 \cdot e^{\varepsilon \cdot |R_{sspace}|},$$

where we define

$$R_{sspace} \overset{\text{def}}{=} E_0|_{sspace,H_0} \Delta E_1|_{sspace,H_0}.$$

We can rewrite the cardinality of $R_{sspace}$ and thus know that

$$p_0 \leq p_1 \cdot e^{\sum_{r \in R_{sspace}} \varepsilon}.$$

By definition of $H_0'$ we know that $H_0'(r) - H_0(r) = \varepsilon$ for all $r \in \mathcal{R}$ such that $sspace(r, H_0(r))$ and 0 otherwise. Thus,

$$p_0 \leq p_1 \cdot e^{\sum_{r \in R_{sspace}} H_0'(r) - H_0(r)}.$$

We know that

$$R_{sspace} \overset{\text{def}}{=} E_0|_{sspace,H_0} \Delta E_1|_{sspace,H_0}.$$

We also know that by definition

$$E_i|_{sspace,H_0} = \{r \in E_i \mid sspace(r, H_0(r))\} = E_i \cap \{r \in \mathcal{R} \mid sspace(r, H_0(r))\}.$$

By statement 1 of Lemma 1 it follows that

$$R_{sspace} \overset{\text{def}}{=} E_0|_{sspace,H_0} \Delta E_1|_{sspace,H_0} = (E_0 \Delta E_1)|_{sspace,H_0} = R|_{sspace,H_0}.$$

We know by their definition that

$$R|_{sspace,H_0} \subseteq R \subseteq R_\alpha \cup R.$$

We can thus conclude that

$$p_0 \leq p_1 \cdot e^{\sum_{r \in R_{sspace}} H_0'(r) - H_0(r)} \leq p_1 \cdot e^{\sum_{r \in R_\alpha \cup R} H_0'(r) - H_0(r)}.$$

Thus, (C.3) is fulfilled.

**Case: Probabilistic query rule with silent record dropping (Query-Drop).** By the definition of *dist* we know that there must exist some value $n \in \mathsf{Val}$ such that $\alpha_0 = \alpha_1 = n$ and $R_\alpha = \mathit{dist}(\alpha_0, \alpha_1) = \emptyset$. Using our previous knowledge and the definition of (Query-Drop) we can immediately observe that $P'_0 = P'_1$, $H'_0 = H'_1$, $T'_0 = T'_1$ and deduce (C.1). We also know that $E'_i = E_i$ for $i \in \{0, 1\}$ and thus $E'_0 \Delta E'_1 = E_0 \Delta E_1 \subseteq R \subseteq R_\alpha \cup R$ (C.2). We know that for all $r \in \mathcal{R}$ the history $H'_0 = H'_1$ is defined as $H'_0(r) \overset{\text{def}}{=} H_0(r) + \varepsilon$ if $\mathit{sspace}(r, H_0(r))$ and $H_0(r) + \varepsilon \leq r.c_B$ and as $H'_0(r) \overset{\text{def}}{=} H_0(r)$ otherwise. Using (P.5) we can directly deduce that $H'_0(r) \leq r.c_B$, for all $r \in \mathcal{R}$ (C.4). The probabilities $p_i$ are defined as

$$\mathrm{Prob}[Q^{\mathsf{drop}}_\varepsilon(E_i\|_{\mathit{sspace}, H_0, \varepsilon}) = n].$$

By the differential privacy of the underlying query mechanism $Q^{\mathsf{drop}}_\varepsilon(sv)$ it must thus be the case that

$$p_0 \leq p_1 \cdot e^{\varepsilon \cdot |R_{\mathit{sspace}}|},$$

where we define

$$R_{\mathit{sspace}} \overset{\text{def}}{=} E_0\|_{\mathit{sspace}, H_0, \varepsilon} \Delta E_1\|_{\mathit{sspace}, H_0, \varepsilon}.$$

We can rewrite the cardinality of $R_{\mathit{sspace}}$ and thus know that

$$p_0 \leq p_1 \cdot e^{\sum_{r \in R_{\mathit{sspace}}} \varepsilon}.$$

We also know that by definition

$$\begin{aligned} E_i\|_{\mathit{sspace}, H_0, \varepsilon} &= \{r \in E_i \mid \mathit{sspace}(r, H_0(r)) \wedge H_0(r) + \varepsilon \leq r.c_B\} \\ &= E_i \cap \{r \in \mathcal{R} \mid \mathit{sspace}(r, H_0(r)) \wedge H_0(r) + \varepsilon \leq r.c_B\}. \end{aligned}$$

We can thus apply statement 1 of Lemma 1 to deduce that

$$\begin{aligned} R_{\mathit{sspace}} &\overset{\text{def}}{=} E_0\|_{\mathit{sspace}, H_0, \varepsilon} \Delta E_1\|_{\mathit{sspace}, H_0, \varepsilon} \\ &= (E_0 \Delta E_1)\|_{\mathit{sspace}, H_0, \varepsilon} \\ &= R\|_{\mathit{sspace}, H_0, \varepsilon}. \end{aligned}$$

By definition of $H'_0$ we know that $H'_0(r) - H_0(r) = \varepsilon$ for all $r \in \mathcal{R}$ such that both $sspace(r, H_0(r))$ and $H_0(r) + \varepsilon \leq r.c_B$ and 0 otherwise. Thus, by the definition of $H'_0$ and $R\|_{sspace,H_0,\varepsilon}$ it holds that $H'_0(r) - H_0(r) = \varepsilon$ for all $r \in R\|_{sspace,H_0,\varepsilon}$ and hence

$$\begin{aligned} p_0 &\leq p_1 \cdot e^{\sum_{r \in R_{sspace}} \varepsilon} \\ &= p_1 \cdot e^{\sum_{r \in R\|_{sspace,H_0,\varepsilon}} \varepsilon} \\ &= p_1 \cdot e^{\sum_{r \in R\|_{sspace,H_0,\varepsilon}} H'_0(r) - H_0(r)}. \end{aligned}$$

We know by their definition that

$$R\|_{sspace,H_0,\varepsilon} \subseteq R \subseteq R_\alpha \cup R.$$

Since $H'_0(r) \geq H_0(r)$ for all $r$, we can conclude that

$$p_0 \leq p_1 \cdot e^{\sum_{r \in R_\alpha \cup R} H'_0(r) - H_0(r)}.$$

Thus, (C.3) is fulfilled. □

**Theorem 3** (Configuration invariant (traces)). *Let* $\mathbb{C}_0 = (P_0, E_0, H_0, T_0)$, $\mathbb{C}'_0 = (P'_0, E'_0, H'_0, T'_0)$, $\mathbb{C}_1 = (P_1, E_1, H_1, T_1)$, $\mathbb{C}'_1 = (P'_1, E'_1, H'_1, T'_1) \in \mathsf{Config}$ *be four configurations,* $\sigma_0, \sigma_1 \in \mathsf{Act}^*$ *two traces,* $p_0, p_1 \in [0,1]$ *two probabilities, and* $R_\sigma, R \in 2^{\mathcal{R}}$ *two sets of records.*

*If*

*(P.1) for* $i \in \{0,1\}$ *it holds that* $\mathbb{C}_i \overset{\sigma_i}{\Longrightarrow}_{p_i} \mathbb{C}'_i$ *and*

*(P.2)* $P_0 = P_1$ *and* $H_0 = H_1$ *and* $T_0 = T_1$ *and*

*(P.3)* $dist(\sigma_0, \sigma_1) = R_\sigma$ *and*

*(P.4)* $E_0 \Delta E_1 = R$ *and*

*(P.5) for all* $r \in \mathcal{R}$ *it holds that* $H_0(r) \leq r.c_B$

*then*

*(C.1)* $P'_0 = P'_1$ *and* $H'_0 = H'_1$ *and* $T'_0 = T'_1$ *and*

*(C.2)* $E'_0 \Delta E'_1 \subseteq R_\sigma \cup R$ *and*

*(C.3)* $p_0 \leq p_1 \cdot e^{\sum_{r \in R_\sigma \cup R} H'_0(r) - H_0(r)}$ *and*

*(C.4) for all* $r \in \mathcal{R}$ *it holds that* $H'_0(r) \leq r.c_B$.

*Proof.* We define $m \in \mathbb{N}$ as the length of the two traces $\sigma_i$, i.e.,, $m \overset{\text{def}}{=} |\sigma_0\ | = |\sigma_1|$. The proof proceeds by induction on $m$.

**Case:** $m = 0$. In this base case it immediately follows that $p_0 = p_1 = 1$, $\sigma_0 = \sigma_1 = []$, and that $\mathbb{C}_i = \mathbb{C}'_i$ for $i \in \{1, 2\}$. (C.1), (C.2), (C.4) follow immediately from the assumptions. Furthermore, for all $r \in \mathcal{R}$ it holds that $H'_0(r) - H_0(r) = 0$ and thus we can immediately conclude (C.3).

**Case:** $m > 0$. By the definition of trace semantics we know that for $i \in \{0, 1\}$ there must exist two configurations $\mathbb{C}_i = (P''_i, E''_i, H''_i, T''_i) \in \mathsf{Config}$, two labels $\alpha_1 \in \mathsf{Act}$, two traces $\sigma'_i \in \mathsf{Act}^*$ and four probabilities $p'_i, p''_i \in [0, 1]$ such that

- $\sigma_i = \alpha_i \sigma'_i$ and
- $p_i = p'_i \cdot p''_i$ and
- $\mathsf{C}_i \xrightarrow{\alpha_i}_{p'_i} \mathsf{C}''_i$ and
- $\mathsf{C}''_i \overset{\sigma'_i}{\Longrightarrow}_{p''_i} \mathsf{C}'_i$.

We define $R_\alpha \in 2^{\mathcal{R}}$ as $dist(\alpha_0, \alpha_1)$. We can immediately apply Theorem 2 to deduce that

- $P''_0 = P''_1$ and $H''_0 = H''_1$ and $T''_0 = T''_1$ and
- $E''_0 \Delta E''_1 \subseteq R_\alpha \cup R$ and
- $p'_0 \leq p'_1 \cdot e^{\sum_{r \in R_\alpha \cup R} H''_0(r) - H_0(r)}$ and
- for all $r \in \mathcal{R}$ it holds that $H''_0(r) \leq r.c_B$.

We would like to apply the induction hypothesis to $\mathsf{C}''_i \overset{\sigma'_i}{\Longrightarrow}_{p''_i} \mathsf{C}'_i$ (note that $|\sigma'_i| = |\sigma_i| - 1$), using this as (P.1). We have just shown that (P.2) is also fulfilled. For (P.3) and (P.4) we use $R_{\sigma'}, R' \in 2^{\mathcal{R}}$ to denote $R_{\sigma'} = dist(\sigma'_0, \sigma'_1)$ and $R' = E''_0 \Delta E''_1$. We have already previously shown the fact that for all $r \in \mathcal{R}$ it holds that $H''_0(r) \leq r.c_B$ (P.5). We can thus apply the induction hypothesis and derive that

- $P'_0 = P'_1$ and $H'_0 = H'_1$ and $T'_0 = T'_1$ and
- $E'_0 \Delta E'_1 \subseteq R_{\sigma'} \cup R'$ and
- $p''_0 \leq p''_1 \cdot e^{\sum_{r \in R_{\sigma'} \cup R'} H'_0(r) - H''_0(r)}$ and
- for all $r \in \mathcal{R}$ it holds that $H'_0(r) \leq r.c_B$.

We can immediately see that (C.1) and (C.4) hold. Furthermore we know that since $p_0 = p'_i \cdot p''_i$ it must be the case that

$$\begin{aligned} p_i &= p'_0 \cdot p''_0 \leq p'_1 \cdot e^{\sum_{r \in R_\alpha \cup R} H''_0(r) - H_0(r)} \cdot p''_1 \cdot e^{\sum_{r \in R_{\sigma'} \cup R'} H'_0(r) - H''_0(r)} \\ &= p'_1 \cdot p''_1 \cdot e^{\sum_{r \in R_\alpha \cup R} H''_0(r) - H_0(r) + \sum_{r \in R_{\sigma'} \cup R'} H'_0(r) - H''_0(r)}. \end{aligned}$$

We note that by definition of *dist* it holds that $R_\alpha \subseteq R_\sigma$ and thus $R_\alpha \cup R \subseteq R_\sigma \cup R$. Furthermore, by the definition of *dist* it follows that $R_{\sigma'} \subseteq R_\sigma$ and that from our previous results it holds that $R' = E''_0 \Delta E''_1 \subseteq R_\alpha \cup R \subseteq R_\sigma \cup R$. Thus, $R_{\sigma'} \cup R' \subseteq R_\sigma \cup (R_\sigma \cup R) = R_\sigma \cup R$. Since we know that $E'_0 \Delta E'_1 \subseteq R_{\sigma'} \cup R'$ we can immediately deduce $E'_0 \Delta E'_1 \subseteq R_\sigma \cup R$, thus showing (C.2). Furthermore, by Observation A.0.1 we know that for all $r \in \mathcal{R}$ it holds that $H''_0(r) - H_0(r) \geq 0$ and $H'_0(r) - H''_0(r) \geq 0$.

We conclude that

$$\begin{aligned} p_0 &\leq p'_1 \cdot p''_1 \cdot e^{\sum_{r \in R_\alpha \cup R} H''_0(r) - H_0(r) + \sum_{r \in R_{\sigma'} \cup R'} H'_0(r) - H''_0(r)} \\ &\leq p'_1 \cdot p''_1 \cdot e^{\sum_{r \in R_\sigma \cup R} H''_0(r) - H_0(r) + \sum_{r \in R_\sigma \cup R} H'_0(r) - H''_0(r)} \\ &\leq p'_1 \cdot p''_1 \cdot e^{\sum_{r \in R_\sigma \cup R} H''_0(r) - H_0(r) + H'_0(r) - H''_0(r)} \\ &\leq p'_1 \cdot p''_1 \cdot e^{\sum_{r \in R_\sigma \cup R} H'_0(r) - H_0(r)} \\ &= p_1 \cdot e^{\sum_{r \in R_\sigma \cup R} H'_0(r) - H_0(r)}, \end{aligned}$$

thus showing (C.3). □

Using Theorem 3 we show the privacy guarantees of UniTraX (Theorem 1).

**Restatement A.0.1** (Theorem 1: Privacy of UniTraX). *Let* $\mathbb{C} = (P, E, H, T) \in \mathsf{Config}$ *be a configuration with* $H(r) \leq r.c_B$ *for every* $r$, $\sigma_0, \sigma_1 \in \mathsf{Act}^*$ *two traces,* $p_0, p_1 \in [0,1]$ *two probabilities, and* $R \in 2^{\mathcal{R}}$ *a set of records.*

*If*

*(P.1) for* $i \in \{0,1\}$ *it holds that* $\mathbb{C} \overset{\sigma_i}{\Longrightarrow}_{p_i}$ *and*

*(P.2)* $\mathit{dist}(\sigma_0, \sigma_1) = R$

*then* $p_0 \leq p_1 \cdot e^{\sum_{r \in R} r.c_B}$.

*Proof.* We note that $E \Delta E = \emptyset$. We can thus immediately apply Theorem 3. It follows that there exists a configuration $\mathbb{C}' = (P', E', H', T') \in \mathsf{Config}$ such that $\mathbb{C} \overset{\sigma_0}{\Longrightarrow}{}^{m}_{p_0} \mathbb{C}'$ and

- $p_0 \leq p_1 \cdot e^{\sum_{r \in R} H'(r) - H(r)}$ and

- for all $r \in \mathcal{R}$ it holds that $H'(r) \leq r.c_B$.

From the last two statements we deduce that

$$p_0 \leq p_1 \cdot e^{\sum_{r \in R} H'(r) - H(r)} \leq p_1 \cdot e^{\sum_{r \in R} H'(r)} \leq p_1 \cdot e^{\sum_{r \in R} r.c_B}.$$

□

# Bibliography

[1] Jame O. Achugbue and Francis Y. L. Chin. "The Effectiveness of Output Modification by Rounding for Protection of Statistical Data Bases". In: *Infor* 17.3 (1979), pages 209–218.

[2] Nabil R. Adam and John C. Wortmann. "Security-Control Methods for Statistical Databases: A Comparative Study". In: *ACM Computing Surveys (CSUR)* 21.4 (1989), pages 515–556. DOI: 10.1145/76894.76895.

[3] Vangalur S. Alagar. "Range Equations and Range Matrices: a Study in Statistical Database Security". In: *Proceedings of the International Conference on Cryptology (AUSCRYPT'90)*. Edited by Jennifer Seberry and Josef Pieprzyk. Volume 453. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1990, pages 360–385. DOI: 10.1007/BFb0030376.

[4] Vangalur S. Alagar, Bernard Blanchard, and David Glaser. "Effective inference control mechanisms for securing statistical databases". In: *Proceedings of the AFIPS National Computer Conference (AFIPS'81)*. Volume 50. AFIPS Conference Proceedings. New York: ACM, 1981, pages 443–452. DOI: 10.1145/1500412.1500476.

[5] Mohammad Alaggan, Sébastien Gambs, and Anne-Marie Kermarrec. "Heterogeneous Differential Privacy". In: *Journal of Privacy and Confidentiality* 7.2 (2016), pages 127–158. URL: http://repository.cmu.edu/jpc/vol7/iss2/6.

[6] Mário S. Alvim, Miguel E. Andrés, Konstantinos Chatzikokolakis, Pierpaolo Degano, and Catuscia Palamidessi. "Differential Privacy: On the Trade-Off between Utility and Information Leakage". In: *Proceedings of the 8th International Workshop on Formal Aspects of Security and Trust (FAST'11)*. Edited by Gilles Barthe, Anupam Datta, and Sandro Etalle. Volume 7140. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pages 39–54. DOI: 10.1007/978-3-642-29420-4_3.

[7] Jane R. Bambauer, Krish Muralidhar, and Rathindra Sarathy. "Fool's Gold: An Illustrated Critique of Differential Privacy". In: *Vanderbilt Journal of Entertainment & Technology Law* 16.4 (2014), pages 701–755. URL: http://jetlaw.org/?p=22904.

[8] Raef Bassily, Adam Groce, Jonathan Katz, and Adam D. Smith. "Coupled-Worlds Privacy: Exploiting Adversarial Uncertainty in Statistical Data Privacy". In: *Proceedings of the 54th IEEE Symposium on Foundations of Computer Science (FOCS'13)*. IEEE, 2013, pages 439–448. DOI: 10.1109/FOCS.2013.54.

[9] Leland L. Beck. "A Security Mechanism for Statistical Databases". In: *ACM Transactions on Database Systems (TODS)* 5.3 (1980), pages 316–338. DOI: 10.1145/320613.320617.

[10] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'90)*. Edited by Hector Garcia-Molina and H. V. Jagadish. New York: ACM, 1990, pages 322–331. DOI: 10.1145/93597.98741.

[11] Norbert Beckmann and Bernhard Seeger. "A revised R*-tree in comparison with related index structures". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'09)*. Edited by Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul. New York: ACM, 2009, pages 799–812. DOI: 10.1145/1559845.1559929.

[12] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. "The X-tree: An Index Structure for High-Dimensional Data". In: *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB'96)*. Edited by T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda. Morgan Kaufmann, 1996, pages 28–39. URL: http://vldb.org/conf/1996/P028.PDF.

[13] Petr Berka. *PKDD '99 Discovery Challenge – A collaborative effort in knowledge discovery from databases*. University of Economics, Prague. 1999. URL: https://sorry.vse.cz/~berka/challenge/pkdd1999/chall.htm (visited on 12/18/2018).

[14] Raghav Bhaskar, Abhishek Bhowmick, Vipul Goyal, Srivatsan Laxman, and Abhradeep Thakurta. “Noiseless Database Privacy”. In: *Proceedings of the 17th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'11)*. Edited by Dong Hoon Lee and Xiaoyun Wang. Volume 7073. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pages 215–232. DOI: 10.1007/978-3-642-25385-0_12.

[15] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. “Practical privacy: The SuLQ framework”. In: *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'05)*. Edited by Chen Li. New York: ACM, 2005, pages 128–138. DOI: 10.1145/1065167.1065184.

[16] Avrim Blum, Katrina Ligett, and Aaron Roth. “A learning theory approach to non-interactive database privacy”. In: *Proceedings of the 40th ACM Symposium on Theory of Computing (STOC'08)*. Edited by Cynthia Dwork. New York: ACM, 2008, pages 609–618. DOI: 10.1145/1374376.1374464.

[17] Justin Brickell and Vitaly Shmatikov. “The cost of privacy: Destruction of data-mining utility in anonymized data publishing”. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'08)*. Edited by Ying Li, Bing Liu, and Sunita Sarawagi. New York: ACM, 2008, pages 70–78. DOI: 10.1145/1401890.1401904.

[18] T.-H. Hubert Chan, Mingfei Li, Elaine Shi, and Wenchang Xu. “Differentially Private Continual Monitoring of Heavy Hitters from Distributed Streams”. In: *Proceedings of the 12th International Symposium on Privacy Enhancing Technologies (PETS'12)*. Edited by Simone Fischer-Hübner and Matthew K. Wright. Volume 7384. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pages 140–159. DOI: 10.1007/978-3-642-31680-7_8.

[19] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. “Private and Continual Release of Statistics”. In: *ACM Transactions on Information and System Security (TISSEC)* 14.3 (2011), 26:1–26:24. DOI: 10.1145/2043621.2043626.

[20] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. “Privacy-Preserving Stream Aggregation with Fault Tolerance”. In: *Revised Selected Papers of the 16th International Conference on Financial Cryptography and Data Security (FC’12).* Edited by Angelos D. Keromytis. Volume 7397. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pages 200–214. DOI: 10.1007/978-3-642-32946-3_15.

[21] Francis Y. L. Chin. “Security problems on inference control for SUM, MAX, and MIN queries”. In: *Journal of the ACM* 33.3 (1986), pages 451–464. DOI: 10.1145/5925.5928.

[22] Francis Y. L. Chin and Gultekin Özsoyoglu. “Statistical Database Design”. In: *ACM Transactions on Database Systems (TODS)* 6.1 (1981), pages 113–139. DOI: 10.1145/319540.319558.

[23] Francis Y. L. Chin and Gultekin Özsoyoglu. “Auditing and Inference Control in Statistical Databases”. In: *IEEE Transactions on Software Engineering* 8.6 (1982), pages 574–582. DOI: 10.1109/TSE.1982.236161.

[24] Francis Chin and Gultekin Ozsoyoglu. “Auditing for Secure Statistical Databases”. In: *Proceedings of the ACM Annual Conference (ACM’81).* New York: ACM, 1981, pages 53–59. DOI: 10.1145/800175.809832.

[25] European Commission. *2018 reform of EU data protection rules.* 2018. URL: https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en (visited on 01/03/2019).

[26] NYC Taxi & Limousine Commission. *TLC Trip Record Data.* May 2017. URL: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml (visited on 05/19/2017).

[27] U.S. Congress. *Health Insurance Portability and Accountability Act.* U.S. Government Printing Office. 1996. URL: https://www.govinfo.gov/content/pkg/PLAW-104publ191/html/PLAW-104publ191.htm (visited on 01/03/2019).

[28] Tore Dalenius. “A Simple Procedure for Controlled Rounding”. In: *Statistisk Tidskrift.* 3rd series 19 (1981), pages 202–208. URL: http://scb.se/H/Statistisk%20tidskrift%201963-1984/Statistisk-tidskrift-1981.pdf.

[29] Pranav Dandekar, Nadia Fawaz, and Stratis Ioannidis. “Privacy Auctions for Recommender Systems”. In: *Proceedings of the 8th International Workshop on Internet and Network Economics (WINE’12)*. Edited by Paul W. Goldberg and Mingyu Guo. Volume 7695. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pages 309–322. DOI: 10.1007/978-3-642-35311-6_23.

[30] George I. Davida, David J. Linton, C. Russel Szelag, and David L. Wells. “Database Security”. In: *IEEE Transactions on Software Engineering* 4.6 (1978), pages 531–533. DOI: 10.1109/TSE.1978.234140.

[31] Richard A. DeMillo, David P. Dobkin, and Richard J. Lipton. “Even Data Bases That Lie Can Be Compromised”. In: *IEEE Transactions on Software Engineering* 4.1 (1978), pages 73–75. DOI: 10.1109/TSE.1978.231469.

[32] Dorothy E. Denning. “Secure Statistical Databases with Random Sample Queries”. In: *ACM Transactions on Database Systems (TODS)* 5.3 (1980), pages 291–315. DOI: 10.1145/320613.320616.

[33] Dorothy E. Denning, Peter J. Denning, and Mayer D. Schwartz. “The Tracker: A Threat to Statistical Database Security”. In: *ACM Transactions on Database Systems (TODS)* 4.1 (1979), pages 76–96. DOI: 10.1145/320064.320069.

[34] Dorothy E. Denning and Jan Schlörer. “A Fast Procedure for Finding a Tracker in a Statistical Database”. In: *ACM Transactions on Database Systems (TODS)* 5.1 (1980), pages 88–102. DOI: 10.1145/320128.320138.

[35] Dorothy E. Denning and Jan Schlörer. “Inference Controls for Statistical Databases”. In: *Computer* 16.7 (1983), pages 69–82. DOI: 10.1109/MC.1983.1654444.

[36] Irit Dinur and Kobbi Nissim. “Revealing information while preserving privacy”. In: *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS’03)*. Edited by Frank Neven, Catriel Beeri, and Tova Milo. New York: ACM, 2003, pages 202–210. DOI: 10.1145/773153.773173.

[37] David P. Dobkin, Anita K. Jones, and Richard J. Lipton. “Secure Databases: Protection Against User Influence”. In: *ACM Transactions on Database Systems (TODS)* 4.1 (1979), pages 97–106. DOI: 10.1145/320064.320068.

[38] Josep Domingo-Ferrer and Vicenç Torra. “A Critique of $k$-Anonymity and Some of Its Enhancements”. In: *Proceedings of the 3rd International Conference on Availability, Reliability and Security (ARES’08)*. IEEE, 2008, pages 990–993. DOI: 10.1109/ARES.2008.97.

[39] George T. Duncan and Sumitra Mukherjee. “Disclosure Limitation using Autocorrelated Noise”. In: *Database Security, VI: Status and Prospects. Results of the IFIP WG 11.3 Workshop on Database Security (’92)*. Edited by Bhavani M. Thuraisingham and Carl E. Landwehr. Volume A-21. IFIP Transactions. Amsterdam: North-Holland, 1993, pages 211–224.

[40] Cynthia Dwork. “Differential Privacy”. In: *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP’06)*. Edited by Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener. Volume 4052. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pages 1–12. DOI: 10.1007/11787006_1.

[41] Cynthia Dwork. “Differential Privacy: A Survey of Results”. In: *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation (TAMC’08)*. Edited by Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li. Volume 4978. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pages 1–19. DOI: 10.1007/978-3-540-79228-4_1.

[42] Cynthia Dwork. “A firm foundation for private data analysis”. In: *Communications of the ACM* 54.1 (2011), pages 86–95. DOI: 10.1145/1866739.1866758.

[43] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. “Our Data, Ourselves: Privacy Via Distributed Noise Generation”. In: *Proceedings of the 25th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT’06)*. Edited by Serge Vaudenay. Volume 4004. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pages 486–503. DOI: 10.1007/11761679_29.

[44] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. “Calibrating Noise to Sensitivity in Private Data Analysis”. In: *Proceedings of the 3rd Theory of Cryptography Conference (TCC’06)*. Edited by Shai Halevi and Tal Rabin. Volume 3876. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pages 265–284. DOI: 10.1007/11681878_14.

[45] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. "Differential privacy under continual observation". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC'10)*. Edited by Michael Mitzenmacher and Leonard J. Schulman. New York: ACM, 2010, pages 715–724. DOI: 10.1145/1806689.1806787.

[46] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Foundations and Trends in Theoretical Computer Science* 9.3-4 (2014), pages 211–407. DOI: 10.1561/0400000042.

[47] Cynthia Dwork and Adam Smith. "Differential privacy for statistics: What we know and what we want to learn". In: *Journal of Privacy and Confidentiality* 1.2 (2010). URL: http://repository.cmu.edu/jpc/vol1/iss2/2.

[48] Hamid Ebadi, Thibaud Antignac, and David Sands. "Sampling and partitioning for differential privacy". In: *Proceedings of the 14th Conference on Privacy, Security and Trust (PST'16)*. IEEE, 2016, pages 664–673. DOI: 10.1109/PST.2016.7906954.

[49] Hamid Ebadi and David Sands. "Featherweight PINQ". In: *Journal of Privacy and Confidentiality* 7.2 (2016). URL: http://repository.cmu.edu/jpc/vol7/iss2/7.

[50] Hamid Ebadi, David Sands, and Gerardo Schneider. "Differential Privacy: Now it's Getting Personal". In: *Proceedings of the 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'15)*. Edited by Sriram K. Rajamani and David Walker. New York: ACM, 2015, pages 69–81. DOI: 10.1145/2676726.2677005.

[51] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response". In: *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*. Edited by Gail-Joon Ahn, Moti Yung, and Ninghui Li. New York: ACM, 2014, pages 1054–1067. DOI: 10.1145/2660267.2660348.

[52] Lisette Espín-Noboa, Florian Lemmerich, Philipp Singer, and Markus Strohmaier. "Discovering and Characterizing Mobility Patterns in Urban Spaces: A Study of Manhattan Taxi Data". In: *Proceedings of the 25th International Conference on World Wide Web (WWW'16)*. Edited by Jacqueline Bourdeau, Jim Hendler, Roger Nkambou, Ian Horrocks, and Ben Y. Zhao. New York: ACM, 2016, pages 537–542. DOI: 10.1145/2872518.2890468.

[53] I. P. Fellegi and J. J. Phillips. “Statistical Confidentiality: Some Theory and Application to Data Dissemination”. In: *Annals of Economic and Social Measurement* 3.2 (1974), pages 399–409. URL: http://econpapers.repec.org/RePEc:nbr:nberch:10117.

[54] Ulrich Flegel. “Pseudonymizing Unix Log Files”. In: *Proceedings of the International Conference on Infrastructure Security (InfraSec’02)*. Edited by George I. Davida, Yair Frankel, and Owen Rees. Volume 2437. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2002, pages 162–179. DOI: 10.1007/3-540-45831-X_12.

[55] Paul Francis, Sebastian Probst Eide, and Reinhard Munz. “Diffix: High-Utility Database Anonymization”. In: *Proceedings of the 5th Annual Privacy Forum (APF’17)*. Edited by Erich Schweighofer, Herbert Leitold, Andreas Mitrakas, and Kai Rannenberg. Volume 10518. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2017, pages 141–158. DOI: 10.1007/978-3-319-67280-9_8.

[56] Paul Francis, Sebastian Probst Eide, Pawel Obrok, Cristian Berneanu, Sasa Juric, and Reinhard Munz. *Extended Diffix*. 2018. arXiv: 1806.02075 [cs.CR].

[57] Arik Friedman, Izchak Sharfman, Daniel Keren, and Assaf Schuster. “Privacy-Preserving Distributed Stream Monitoring”. In: *Proceedings of the 21st Symposium on Network and Distributed System Security (NDSS’14)*. ISOC, 2014. DOI: 10.14722/ndss.2014.23128.

[58] Johannes Gehrke, Edward Lui, and Rafael Pass. “Towards Privacy for Social Networks: A Zero-Knowledge Based Definition of Privacy”. In: *Proceedings of the 8th Theory of Cryptography Conference (TCC’11)*. Edited by Yuval Ishai. Volume 6597. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pages 432–449. DOI: 10.1007/978-3-642-19571-6_26.

[59] Arpita Ghosh and Aaron Roth. “Selling privacy at auction”. In: *Proceedings of the 12th ACM Conference on Electronic Commerce (EC’11)*. Edited by Yoav Shoham, Yan Chen, and Tim Roughgarden. New York: ACM, 2011, pages 199–208. DOI: 10.1145/1993574.1993605.

[60] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. “Differential Privacy Under Fire”. In: *Proceedings of the 20th USENIX Security Symposium (USS’11)*. USENIX, 2011. URL: http://usenix.org/events/sec11/tech/full_papers/Haeberlen.pdf.

[61] Raquel Hill. “Evaluating the Utility of Differential Privacy: A Use Case Study of a Behavioral Science Dataset”. In: *Medical Data Privacy Handbook*. Edited by Aris Gkoulalas-Divanis and Grigorios Loukides. Berlin, Heidelberg: Springer, 2015, pages 59–82. DOI: 10.1007/978-3-319-23633-9_4.

[62] Raquel Hill, Michael Hansen, Erick Janssen, Stephanie A. Sanders, Julia R. Heiman, and Li Xiong. “A Quantitative Approach for Evaluating the Utility of a Differentially Private Behavioral Science Dataset”. In: *Proceedings of the IEEE International Conference on Healthcare Informatics (ICHI’14)*. IEEE, 2014, pages 276–284. DOI: 10.1109/ICHI.2014.45.

[63] Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C. Pierce, and Aaron Roth. “Differential Privacy: An Economic Method for Choosing Epsilon”. In: *Proceedings of the 27th IEEE Computer Security Foundations Symposium (CSF’14)*. IEEE, 2014, pages 398–410. DOI: 10.1109/CSF.2014.35.

[64] Wiebren de Jonge. “Compromising Statistical Databases Responding to Queries about Means”. In: *ACM Transactions on Database Systems (TODS)* 8.1 (1983), pages 60–80. DOI: 10.1145/319830.319834.

[65] Zach Jorgensen, Ting Yu, and Graham Cormode. “Conservative or liberal? Personalized differential privacy”. In: *Proceedings of the 31st International Conference on Data Engineering (ICDE’15)*. Edited by Johannes Gehrke, Wolfgang Lehner, Kyuseok Shim, Sang Kyun Cha, and Guy M. Lohman. IEEE, 2015, pages 1023–1034. DOI: 10.1109/ICDE.2015.7113353.

[66] Zach Jorgensen, Ting Yu, and Graham Cormode. “Publishing Attributed Social Graphs with Formal Privacy Guarantees”. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD’16)*. Edited by Fatma Özcan, Georgia Koutrika, and Sam Madden. New York: ACM, 2016, pages 107–122. DOI: 10.1145/2882903.2915215.

[67] Krishnaram Kenthapadi, Nina Mishra, and Kobbi Nissim. “Simulatable auditing”. In: *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS’05)*. Edited by Chen Li. ACM, 2005, pages 118–127. DOI: 10.1145/1065167.1065183.

[68] Daniel Kifer and Ashwin Machanavajjhala. “No free lunch in data privacy”. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD’11)*. Edited by Timos K. Sellis, Renée

J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis. New York: ACM, 2011, pages 193–204. DOI: 10.1145/1989323.1989345.

[69] Jon M. Kleinberg, Christos H. Papadimitriou, and Prabhakar Raghavan. “Auditing Boolean Attributes”. In: *Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS’00).* Edited by Victor Vianu and Georg Gottlob. New York: ACM, 2000, pages 86–91. DOI: 10.1145/335168.335210.

[70] Aleksandra Korolova. “Privacy Violations Using Microtargeted Ads: A Case Study”. In: *Proceedings of the 10th IEEE International Conference on Data Mining Workshops (ICDMW’10).* Edited by Wei Fan, Wynne Hsu, Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos, and Xindong Wu. IEEE, 2010, pages 474–482. DOI: 10.1109/ICDMW.2010.137.

[71] Jaewoo Lee and Chris Clifton. “How Much Is Enough? Choosing $\epsilon$ for Differential Privacy”. In: *Proceedings of the 14th International Conference on Information Security (ISC’11).* Edited by Xuejia Lai, Jianying Zhou, and Hui Li. Volume 7001. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pages 325–340. DOI: 10.1007/978-3-642-24861-0_22.

[72] Jaewoo Lee and Chris Clifton. “Differential identifiability”. In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’12).* Edited by Qiang Yang, Deepak Agarwal, and Jian Pei. New York: ACM, 2012, pages 1041–1049. DOI: 10.1145/2339530.2339695.

[73] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. “$t$-Closeness: Privacy Beyond $k$-Anonymity and $l$-Diversity”. In: *Proceedings of the 23rd International Conference on Data Engineering (ICDE’07).* Edited by Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis. IEEE, 2007, pages 106–115. DOI: 10.1109/ICDE.2007.367856.

[74] Tiancheng Li and Ninghui Li. “On the tradeoff between privacy and utility in data publishing”. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’09).* Edited by John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki. New York: ACM, 2009, pages 517–526. DOI: 10.1145/1557019.1557079.

[75] Ashwin Machanavajjhala, Daniel Kifer, John M. Abowd, Johannes Gehrke, and Lars Vilhuber. “Privacy: Theory meets Practice on the Map”. In: *Proceedings of the 24th International Conference on Data Engineering (ICDE’08)*. Edited by Gustavo Alonso, José A. Blakeley, and Arbee L. P. Chen. IEEE, 2008, pages 277–286. DOI: 10.1109/ICDE.2008.4497436.

[76] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. “$l$-Diversity: Privacy beyond $k$-anonymity”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.1, Article 3 (2007). DOI: 10.1145/1217299.1217302.

[77] David J. Martin, Daniel Kifer, Ashwin Machanavajjhala, Johannes Gehrke, and Joseph Y. Halpern. “Worst-Case Background Knowledge for Privacy-Preserving Data Publishing”. In: *Proceedings of the 23rd International Conference on Data Engineering (ICDE’07)*. Edited by Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis. IEEE, 2007, pages 126–135. DOI: 10.1109/ICDE.2007.367858.

[78] David McClure and Jerome P. Reiter. “Differential Privacy and Statistical Disclosure Risk Measures: An Investigation with Binary Synthetic Data”. In: *Transactions on Data Privacy (TDP)* 5.3 (2012), pages 535–552. URL: http://www.tdp.cat/issues11/abs.a093a11.php.

[79] Frank McSherry. *Privacy Integrated Queries (PINQ)*. Microsoft Research. June 2009. URL: https://www.microsoft.com/en-us/research/project/privacy-integrated-queries-pinq/ (visited on 01/02/2019).

[80] Frank McSherry. “Privacy integrated queries: an extensible platform for privacy-preserving data analysis”. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD’09)*. Edited by Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul. New York: ACM, 2009, pages 19–30. DOI: 10.1145/1559845.1559850.

[81] Frank McSherry and Kunal Talwar. “Mechanism Design via Differential Privacy”. In: *Proceedings of the 48th IEEE Symposium on Foundations of Computer Science (FOCS’07)*. IEEE, 2007, pages 94–103. DOI: 10.1109/FOCS.2007.41.

[82] Andrés Monroy-Hernández. *NYC Taxi Trips*. 2014. URL: http://www.andresmh.com/nyctaxitrips/ (visited on 06/26/2014).

[83] Iulian Moraru, David G. Andersen, Michael Kaminsky, Nathan Binkert, Niraj Tolia, Reinhard Munz, and Parthasarathy Ranganathan. *Persistent, Protected and Cached: Building Blocks for Main Memory Data Stores.* Technical report CMU-PDL-11-114. Pittsburgh: Parallel Data Laboratory (PDL), Carnegie Mellon University (CMU), Dec. 2011. URL: http://www.pdl.cmu.edu/PDL-FTP/NVM/CMU-PDL-11-114.pdf.

[84] Iulian Moraru, David G. Andersen, Michael Kaminsky, Nathan Binkert, Niraj Tolia, Reinhard Munz, and Parthasarathy Ranganathan. *Persistent, Protected and Cached: Building Blocks for Main Memory Data Stores.* Technical report CMU-PDL-11-114 v2. Pittsburgh: Parallel Data Laboratory (PDL), Carnegie Mellon University (CMU), Nov. 2012. URL: http://www.pdl.cmu.edu/PDL-FTP/NVM/CMU-PDL-11-114v2.pdf.

[85] Reinhard Munz, Fabienne Eigner, Matteo Maffei, Paul Francis, and Deepak Garg. *UniTraX: Protecting Data Privacy with Discoverable Biases.* Technical report MPI-SWS-2018-001. Kaiserslautern and Saarbrücken: Max Planck Institute for Software Systems (MPI-SWS), Feb. 2018. URL: https://www.mpi-sws.org/tr/2018-001.pdf.

[86] Reinhard Munz, Fabienne Eigner, Matteo Maffei, Paul Francis, and Deepak Garg. "UniTraX: Protecting Data Privacy with Discoverable Biases". In: *Proceedings of the 7th International Conference on Principles of Security and Trust (POST'18).* Edited by Lujo Bauer and Ralf Küsters. Volume 10804. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2018, pages 278–299. DOI: 10.1007/978-3-319-89722-6_12.

[87] M. S. Nargundkar and W. Saveland. "Random Rounding to Prevent Statistical Disclosure". In: *Proceedings of the Social Statistics Section (SSS'72).* Edited by Edwin D. Goldfield. American Statistical Association, 1972, pages 382–385. URL: http://amstat.org/sections/srms/Proceedings/y1972/Random%20-Rounding%20To%20Prevent%20Statistical%20Disclosures.pdf.

[88] Mehmet Ercan Nergiz, Maurizio Atzori, and Chris Clifton. "Hiding the presence of individuals from shared databases". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'07).* Edited by Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou. New York: ACM, 2007, pages 665–676. DOI: 10.1145/1247480.1247554.

[89] Thomas Neubauer and Johannes Heurix. “A methodology for the pseudonymization of medical data”. In: *International Journal of Medical Informatics* 80.3 (2011), pages 190–204. DOI: 10.1016/j.ijmedinf.2010.10.016.

[90] Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. “Smooth sensitivity and sampling in private data analysis”. In: *Proceedings of the 39th ACM Symposium on Theory of Computing (STOC'07)*. Edited by David S. Johnson and Uriel Feige. New York: ACM, 2007, pages 75–84. DOI: 10.1145/1250790.1250803.

[91] Kobbi Nissim, Salil P. Vadhan, and David Xiao. “Redrawing the boundaries on purchasing data from privacy-sensitive individuals”. In: *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science (ITCS'14)*. Edited by Moni Naor. New York: ACM, 2014, pages 411–422. DOI: 10.1145/2554797.2554835.

[92] Gultekin Özsoyoglu and Z. Meral Özsoyoglu. “Update Handling Techniques in Statistical Databases”. In: *Proceedings of the 1st LBL Workshop on Statistical Database Management (SSDBM'81)*. Edited by Harry K. T. Wong. Berkeley: Lawrence Berkeley Laboratory, 1982, pages 249–283.

[93] Andreas Pfitzmann and Marit Köhntopp. “Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology”. In: *Proceedings of the International Workshop on Design Issues in Anonymity and Unobservability*. Edited by Hannes Federrath. Volume 2009. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pages 1–9. DOI: 10.1007/3-540-44702-4_1.

[94] Davide Proserpio, Sharon Goldberg, and Frank McSherry. “Calibrating Data to Sensitivity in Private Data Analysis”. In: *PVLDB* 7.8 (2014), pages 637–648. URL: http://www.vldb.org/pvldb/vol7/p637-proserpio.pdf.

[95] Do Le Quoc, Martin Beck, Pramod Bhatotia, Ruichuan Chen, Christof Fetzer, and Thorsten Strufe. “PrivApprox: Privacy-Preserving Stream Analytics”. In: *Proceedings of the USENIX Annual Technical Conference (ATC'17)*. Edited by Dilma Da Silva and Bryan Ford. USENIX, 2017, pages 659–672. URL: https://www.usenix.org/conference/atc17/technical-sessions/presentation/quoc.

[96] Vibhor Rastogi and Suman Nath. "Differentially private aggregation of distributed time-series with transformation and encryption". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'10)*. Edited by Ahmed K. Elmagarmid and Divyakant Agrawal. New York: ACM, 2010, pages 735–746. DOI: 10.1145/1807167.1807247.

[97] Bernhard Riedl, Thomas Neubauer, Gernot Goluch, Oswald Boehm, Gert Reinauer, and Alexander Krumboeck. "A secure architecture for the pseudonymization of medical data". In: *Proceedings of the 2nd International Conference on Availability, Reliability and Security (ARES'07)*. IEEE, 2007, pages 318–324. DOI: 10.1109/ARES.2007.22.

[98] Jan Schlörer. *On statistical confidentiality: Minimum query language requirements for tracker building*. Materialien Nr. 33. Department of Medical Statistics, Documentation and Data Processing, University of Ulm, July 1975.

[99] Jan Schlörer. *Security of Statistical Databases: Ranges and Trackers*. Klinische Dokumentation, Universität Ulm, W. Germany, Nov. 1981.

[100] Jan Schlörer. "Sicherung statistischer Datenbanken: Output von Intervallen". In: *Proceedings of the 3rd Conference of the European Cooperation in Informatics (ECI'81)*. Edited by Wilfried Brauer. Volume 50. Informatik-Fachberichte. Berlin, Heidelberg: Springer, 1981, pages 327–336. DOI: 10.1007/978-3-662-01089-1_30.

[101] Jan Schlörer. "Outputkontrollen zur Sicherung statistischer Datenbanken. Ein Überblick". In: *Informatik Spektrum* 5.4 (1982), pages 224–236.

[102] Jan Schlörer. *Sicherheit statistischer Datenbanken; Untersuchungen zum Identifikations- und Outputproblem; Schlußbericht an die Stiftung Volkswagenwerk*. Klinische Dokumentation, Klinikum der Universität Ulm, Ulm, Germany, July 1984.

[103] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. "Privacy-Preserving Aggregation of Time-Series Data". In: *Proceedings of the Symposium on Network and Distributed System Security (NDSS'11)*. ISOC. 2011. URL: https://www.isoc.org/isoc/conferences/ndss/11/pdf/9_3.pdf.

[104] George L. Sicherman, Wiebren de Jonge, and Reind P. van de Riet. "Answering Queries Without Revealing Secrets". In: *ACM Transactions on Database Systems (TODS)* 8.1 (1983), pages 41–59. DOI: 10.1145/319830.319833.

[105] Andrew C. Simpson, David J. Power, and Mark Slaymaker. "On tracker attacks in health grids". In: *Proceedings of the ACM Symposium on Applied Computing (SAC'06)*. Edited by Hisham Haddad. New York: ACM, 2006, pages 209–216. DOI: 10.1145/1141277.1141326.

[106] Michael Stonebraker. "Implementation of Integrity Constraints and Views by Query Modification". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD'75)*. Edited by W. Frank King. ACM, 1975, pages 65–78. DOI: 10.1145/500080.500091.

[107] Michael Stonebraker and Eugene Wong. "Access Control in a Relational Data Base Management System by Query Modification". In: *Proceedings of the ACM Annual Conference (ACM'74)*. Volume 1. New York: ACM, 1974, pages 180–186. DOI: 10.1145/800182.810400.

[108] Latanya Sweeney. "Datafly: A System for Providing Anonymity in Medical Data". In: *Database Securty XI: Status and Prospects, IFIP TC11 WG11.3 Eleventh International Conference on Database Security ('97)*. Edited by Tsau Young Lin and Shelly Qian. Volume 113. IFIP Conference Proceedings. Chapman & Hall, 1998, pages 356–381.

[109] Latanya Sweeney. "$k$-Anonymity: A Model for Protecting Privacy". In: *International Journal on Uncertainty, Fuzziness and Knowledge-Based Systems* 10.5 (2002), pages 557–570. DOI: 10.1142/S0218488502001648.

[110] Latanya Sweeney. "Achieving k-Anonymity Privacy Protection Using Generalization and Suppression". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.5 (2002), pages 571–588. DOI: 10.1142/S021848850200165X.

[111] Latanya Sweeney. *Only You, Your Doctor, and Many Others May Know.* Technology Science. Sept. 2015. URL: https://techscience.org/a/2015092903 (visited on 01/04/2019).

[112] Latanya Sweeney, Ji Su Yoo, Laura Perovich, Katherine E. Boronow, Phil Brown, and Julia Green Brody. *Re-identification Risks in HIPAA Safe Harbor Data: A study of data from one environmental health study.* Tech-

nology Science. Aug. 2017. URL: https://techscience.org/a/2017082801 (visited on 01/04/2019).

[113] Aron Szanto and Neel Mehta. *A Host of Troubles: Re-Identifying Airbnb Hosts Using Public Data.* Technology Science. Oct. 2018. URL: https://techscience.org/a/2018100902 (visited on 01/04/2019).

[114] Patrick Tendick and Norman S. Matloff. "A Modified Random Perturbation Method for Database Security". In: *ACM Transactions on Database Systems (TODS)* 19.1 (1994), pages 47–63. DOI: 10.1145/174638.174641.

[115] J.K. Trotter. *Public NYC Taxicab Database Lets You See How Celebrities Tip.* Gawker. Oct. 2014. URL: http://gawker.com/the-public-nyc-taxicab-database-that-accidentally-track-1646724546 (visited on 01/22/2016).

[116] Ji Su Yoo, Alexandra Thaler, Latanya Sweeney, and Jinyan Zang. *Risks to Patient Privacy: A Re-identification of Patients in Maine and Vermont Statewide Hospital Data.* Technology Science. Oct. 2018. URL: https://techscience.org/a/2018100901 (visited on 01/04/2019).

[117] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. "DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language". In: *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI'08).* Edited by Richard Draves and Robbert van Renesse. USENIX, 2008, pages 1–14. URL: https://www.usenix.org/event/osdi08/tech/full_papers/yu_y/yu_y.pdf.

[118] Simon Van Zuylen-Wood. *The Struggles of New York City's Taxi King.* Aug. 2015. URL: http://www.bloomberg.com/features/2015-taxi-medallion-king/ (visited on 08/02/2016).