Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bachelor's Thesis Nr. 361b

Systems Group, Department of Computer Science, ETH Zurich

Privacy Transformation Planner for User-centric Data Privacy Systems

by

Emanuel R. Opel

Supervised by

Nicolas Küchler, Lukas Burkhalter, Dr. Anwar Hithnawi, Prof. Timothy Roscoe

13.03.2021 - 13.09.2021

Abstract

In recent years, we have witnessed a significant increase in interest and adoption of strict data privacy controls across organizations. This development is primarily attributed to a general shift in public perception of privacy and the pervasiveness of data misuse and breaches. New data privacy systems are emerging and progressively find their way into existing data systems.

These privacy developments raise a new fundamental challenge in existing data systems, namely privacy management. Privacy presents some unique constraints, and this includes dealing with a heterogeneous set of privacy policies (in all user-centric systems and, e.g., due to different jurisdictions) and managing limited, non-replenishable resources (what we call a ”privacy budget”). Hence, this necessitates new solutions that are tailored to privacy.

In this thesis, we address the following challenges in the privacy management lifecycle:

- Designing generic privacy controls and mapping them to a set of allowed privacy transformations based on data type and application.
- Developing methods to manage privacy resources that arise from associated privacy controls.

In this work, we propose a privacy management system that offers controls for both users and service providers to design privacy policies. As different parts of data are tagged with different privacy controls, the issue of how this data can still be combined arises. We want to serve queries operating on this data to respect these privacy controls and ensure that privacy resources are optimized for the longevity of data use. In other words, we want to serve as many queries as possible (perhaps with some prioritization among those queries, e.g., if some queries are more important than others) while still respecting all privacy constraints.

This task boils down to a matching problem: matching queries and data subjects in a way that respects all constraints and optimizes a target function. Towards solving this problem, we present a unified model that integrates all privacy constraints and objectives regarding data usage. Using this model, we present a greedy, heuristic approach that quickly produces good assignments with low memory usage. However, for more complicated settings, this approach may suffer from suboptimal management of privacy resources. We then introduce an integer linear program (ILP) approach to address this issue, which allows finding near-optimal solutions (with a modifiable bound on suboptimality, e.g., 1% or 0.1%) for scenarios where latency and memory requirements are of less concern. Produced assignments (by

either approach) can then be executed to obtain a policy-compliant view of the data.

For a nontrivial example setup, we show that both the heuristic and the ILP approach scale linearly in both runtime and memory usage regarding the number of users, even though ILP programs are widely known to scale exponentially in the worst case. Still, the ILP approach is more costly in terms of memory and runtime. However, we show that the ILP approach can significantly outperform the heuristic approach by better leveraging limited privacy resources to more closely conform to the providers' preferences. Finally, we show that in particular situations, the heuristic algorithm also produces near-optimal results. However, the more complex the set of privacy policies and queries becomes, the more significant the comparative advantage of the ILP approach.

Acknowledgements

Since starting this thesis, I have been supported in many different ways by many different people.

First and foremost, I would like to extend my deepest gratitude to my supervisors Nicolas Küchler, Lukas Burkhalter, and Dr. Anwar Hithnawi. With Nicolas, I regularly had long discussions about many technical and mathematical aspects of the work. In addition, he came up with many key ideas that shaped the whole direction of this thesis and helped me throughout all stages of the more practical part of this thesis. In particular, he was very patient with the many difficulties I faced towards the end of the thesis, answered numerous questions I had during this time, and gave regular, constructive feedback regarding this write-up. Lukas shaped many design decisions and made sure numerous practical problems arising from different ideas did not go unnoticed. His skill of explaining complex problems and ideas in simple and intuitive ways was of great help on numerous occasions. Anwar's guidance was of particular importance for the write-up. I received much constructive feedback from her, which compensated for my lack of experience regarding writing a thesis. Her regular hints regarding related work and essential discussions about the bigger picture were also of great importance to guide the direction of this work.

I had regular, long meetings with all my supervisors where we discussed various ideas, and I got a lot of constructive criticism and suggestions. Even though, due to COVID-19, our interactions were mostly limited to online meetings and discussions over slack, I got valuable insights and skills relating to the whole process of writing a thesis, from the initial ideas and design stages over the implementation parts up to writing down everything.

Next, I am very grateful to Dr. Remigius Meier, who read through many parts of this thesis and provided invaluable feedback from a different perspective.

On top of this, I am very grateful to Prof. Timothy Roscoe, for allowing me to work in this fantastic environment with very skilled and motivated people. Furthermore, I would also like to thank Alexander Viand for his helpful advice regarding the write-up in general and specific questions that arose at some points.

Contents

1 Introduction

In recent years, the collection and processing of sensitive data have reached new heights. Applications across the spectrum rely on users' data to power their essential services. For example, consider Google Maps: Accurate and timely traffic tools are critical to the navigation app and are only possible thanks to users sharing data [15]. More generally, data-driven services are increasingly deployed in practice [32].

However, with rapidly growing amounts of collected data, misuse of data is on the rise [9]. Since much of the involved data is inherently sensitive (e.g., credit scores), public awareness of data privacy has dramatically shifted.

These developments have led to privacy and security gaining tremendous importance across almost all organizations. Any organization not handling such issues with care may face substantial reputational risks. On top of that, regulatory demands and oversight have increased, translating to legal risk for organizations with inadequate data security and privacy practices. On the other hand, good privacy practices are now actively being used as advertisement, for example, by Apple [26].

Thus, it comes as no surprise that recently, a flurry of privacy systems and solutions emerged. However, many of these solutions are custom-made for specific applications or use cases. However, also some designs for end-to-end privacy systems emerged that aim at targeting broad application scenarios (e.g., Privitar [27], or Zeph [4]).

Ensuring data privacy has profound consequences on every aspect of data collection and processing. First, we need to protect data from unauthorized users. This task can be handled by encryption and access control. Second, a sound privacy design requires ensuring data minimization and purpose limitation by design, even for authorized users. Privacy solutions handle these objectives. However, appropriate mechanisms and designs may vary widely depending on applications and data type, which is where the complexity of generic privacy solutions arises.

In particular, the type and form of data collected, how this data is managed, and which systems are used in this process are central pieces determining which kinds of privacy and security guarantees any application or process can offer. For example, if some data is not collected in the first place, an application trivially cannot leak that data. In principle, any insecure piece that plays some part in data management and processing could void any privacy and security guarantees. These problems can be partly overcome using cryptographic methods, but not without a cost. We, therefore, argue that end-to-end privacy tools are invaluable to achieve better privacy practices on an application level.

As a first step towards better data privacy solutions, we need to organize and manage data on a conceptual level to ensure users' privacy preferences are respected. On a lower level, this means managing privacy resources (such as which data may be reused or how much privacy budget may be used). Of course, we need to respect constraints that we are trusted to uphold by the user while also ensuring maximum possible utility for any analysts or similar trying to generate value from data. Not sharing any data certainly gives maximum possible privacy protection for the user but simultaneously inhibits all beneficial data usage for an organization or society.

1.1 Challenges and Approach

This thesis first sheds some light on the implications and challenges of data management systems when considering privacy as a critical constraint.

The first challenge consists of defining an expressive, coherent, and practical schema to define privacy policies and a set of queries that should be executed on the data. The schema for privacy policies should allow for a balanced control between users and providers while still incorporating a wide range of privacy policies different users may have. Similarly, the application provider (or analysts) may want to express a wide range of queries, which the framework needs to support.

The second significant challenge is reconciling queries with a possibly heterogeneous set of privacy policies, i.e., designing the privacy planner. Of course, this is connected to the first challenge since more expressive schemas for privacy policies and queries might make this reconciliation harder. For example, we need to provide a way for the provider to express their preferences regarding which queries should be prioritized. The privacy planner should then match users' data streams and queries, such that the solution complies with all privacy policies while conforming to the providers' preferences as much as possible. On top of this, we need to consider how different designs would scale in an actual implementation.

Finally, implementing a transformation planner constitutes the third challenge. We need to carefully consider the tradeoff between the performance of the transformation planner in terms of hardware utilization and the optimality of found solutions. System optimizations and some modifications of the underlying approach can go a long way towards finding more favorable tradeoffs in this regard. Of particular concern is the system's scalability, as modern applications may have millions of users.

We propose an end-to-end privacy solution that is general enough to suit a wide range of applications and user demands regarding data privacy. To this end, we allow users to define their privacy policy according to their preferences from a range of options chosen by the application provider. Furthermore, the provider or analysts may express a wide range of desired queries and preferences regarding which queries should be prioritized.

Towards reconciling these diverging interests, we present two fundamentally different approaches towards matching queries with user data (i.e., the privacy planner). The privacy planner respects all user-defined privacy policies while taking the provider's preferences into accounts as much as possible. The approaches feature different tradeoffs concerning optimality (regarding the specified priority of queries) and system performance (in terms of hardware usage). These approaches are not limited to our framework and can be adapted by other privacy frameworks that adopt a user-centric approach and generalize to various applications.

1.2 Contributions

This thesis presents the following contributions:

- We recognize common privacy transformations and privacy models and integrate them into a unified abstraction. We expose an API to the user, allowing simple controls (within limits specified by the provider) to specify constraints regarding allowed transformations on their data, which are automatically mapped to constraints in the abstract model. Another API is available to the provider, which allows specifying a wide range of queries and priorities on these queries. These are then automatically mapped into our abstract model as target objectives.
- We propose two solutions to reconcile constraints set by users' privacy policies and the demands expressed in queries. For time- and memory-constrained settings, we propose a greedy, heuristic algorithm. This approach, however, comes with the tradeoff of not guaranteeing any bound regarding the optimality of the solution. Where latency and memory requirements are of less concern, we propose an approach using ILP solvers to find optimal solutions up to a specific, modifiable bound (e.g., 1%).
- Finally, we implement these two approaches in a prototype. We then evaluate this prototype with a focus on scalability and tradeoffs between the approaches under various settings.

1.3 Thesis Outline

We start this thesis with an overview of some necessary background material in Section 2. The covered material includes an overview of some definitions for data privacy, current privacy regulations, and privacy-enhancing technologies. With this background knowledge, we then discuss related work in the area of privacy that is relevant to this thesis in Section 3.

The core of the thesis is formed by our proposed solutions to the challenges mentioned above. In Section 4, we give an overview of our design. We then go into details regarding the framework for privacy policies and queries, as well as the transformation planner. Finally, in Section 5, we present an overview of our prototype, and evaluate it in Section 6, before drawing a conclusion in Section 7.

2 Background

In this section, we cover background material relevant to the thesis. We start with a brief introduction to data privacy. Then we discuss key technological concepts of central importance to understanding the content of this thesis and related work.

2.1 Data Privacy

The right to privacy as a legal concept was first introduced [14] in a law review article from 1890 where the right to privacy was described as the right to be let alone [37]. In other words, privacy was regarded as being left alone, to remain unobserved. While this definition has not lost its validity, to remain unobserved is today probably best described by the word ”secrecy”.

Privacy as a concept was established before the digital world as we know it today came to be. However, with the emergence and development of the Internet, the notion of digital privacy has evolved and is gaining tremendous importance. The Internet and most other digital inventions have emerged without much consideration to the right to privacy (at least regarding the above definition). This is largely due to the excitement around big data and its potential benefits, which often overshadowed its potential pitfalls and implications on society, particularly concerning individuals’ right to privacy. The concept of privacy seems to work in the opposite direction. Keeping data secret implies that many applications cannot reach their full potential.

Therefore, secrecy (and with it privacy) seems to be fundamentally incompatible with the modern digital world that is largely data driven. However, just looking at whether or not data was kept secret seems hardly enough to determine the impact on individuals. For example, when data is processed in aggregate forms, the impact on an individual is likely lower than if each individual’s data is examined individually.

With this, alternative definitions of privacy emerged to strike a balance between preserving individuals’ privacy and continuing to benefit from users’ data. In other words, the focus is on the impact of sharing data with all the different actors and subjects. For example, Prof. Nissenbaum defines privacy as the ”appropriate flow of information” [25] in the context of her contextual integrity theory.

The above and similar definitions directly imply that solutions limiting what can be inferred from data (purpose limitations) and only collecting strictly necessary data (data minimization) are of central importance to preserve privacy. We argue that just keeping all data secret is not the answer to the issue of privacy, as all data subjects are also users, which can benefit from certain kinds of data processing.

In this spirit, many solutions have emerged that allow a much more fine-grained control over data, based on these two core pillars of data minimization and purpose limitation.

2.1.1 Privacy Regulation

With the advances in data collection and processing, privacy concerns are continuously gaining importance. Lawmakers in many jurisdictions have acted upon this and adopted comprehensive privacy regulations, like the EU with the General Data Protection Regulation (GDPR) or Switzerland with the Federal Act on Data Protection (FADP). This linkage of rising privacy concerns with new inventions was already present in 1890 in the US, where the influential article ”The Right to Privacy” [37] was published in the Harvard law review. The authors explicitly justified the need for privacy with (among other reasons) ”Instantaneous photographs and newspaper enterprise (...) and numerous mechanical devices” that threatened to expose everyone’s hidden secrets to the world.

Even though the regulations may look very different in different jurisdictions, some significant trends can be identified. Most importantly, many regulations require services to inform a user how their data can and will be used (e.g., FADP [10] article 4 or GDPR [13] article 5). For this reason, commercial service providers usually have a privacy notice or privacy policy in some form, which the user is made aware of when signing up for a service. While consent is not always required (see, e.g., FADP articles 12, 13 or GDPR [13] article 6), it is often still a prerequisite to sign up for services as a legal safeguard for the provider.

Implementation. To implement these privacy regulations, developers usually need to work together with legal experts who then draw up an appropriate privacy policy. In essence, developers need to provide a list of use cases to the legal experts to then integrate this into a services privacy notice/privacy policy. For many types of services, more or less standardized privacy notices/policies exist. Still, for novel ways of data usage, developers may need to consult with legal experts.

Enforcement. Enforcing privacy policies in today’s systems is currently mainly a manual task. However, there are efforts to automate this process (e.g., Ancile, Multiverse Databases, and Qapla [3, 21, 23]). Without automatic enforcement, users have to trust the developers and legal experts of a service that a privacy policy accurately describes how the data is used. Without proper enforcement, this can leave data vulnerable to privacy violations. Work on automatic verification and validation is essential to try to overcome this issue.

While the predominant ”inform and consent” approach adopted in many privacy regulations is a big step towards more transparency in data usage, it does not give users much control over their data. Even when a service has many privacy options, these are usually just binary switches where data usages for specific purposes (e.g., personalized advertisements) can be authorized. How the data is used for these purposes is often not immediately apparent.

Therefore, the next step is to enable more comprehensive privacy options that can more accurately model a user’s privacy preferences regarding different applications. Such an approach can lead to more data being voluntarily shared with the provider since users do not need to deny all data sharing if they only want to restrict some kinds of usages. For example, users might be comfortable sharing their data if it is only used in aggregation with a large enough group of other users. In summary, such an approach can allow for the safe use of data while still respecting users’ privacy preferences.

2.2 Privacy-Enhancing Technologies

In this section, various concepts that can be used to enhance privacy or model and measure privacy are introduced, if they are either relevant for this thesis or indispensable for related work.

2.2.1 Privacy Models

Privacy models try to formalize the notion of privacy. Through that, they often assign concrete numerical values for the amount of privacy upheld or lost by certain kinds of data usage.

Differential Privacy

Differentially privacy tries to capture the privacy impact of executing randomized algorithms that take as input a collection of data corresponding to different persons. The idea is to add random noise to the output of such an algorithm. The contribution of a single person to the algorithm’s output is then probabilistically similar to the algorithm’s output if this person’s data was not present in the algorithm dataset. The variable ϵ captures the ”probabilistic similarity”. For small values of ϵ, little information about this person is revealed, therefore protecting the data subject’s privacy. Formally, differential privacy is defined as follows ([38], Definition 2.1):

Definition 2.1 (Differential Privacy)**.** *A randomized Algorithm M provides ϵ-differential privacy if for any two adjacent datasets A and B and any set of possible outputs S of M:*

$$\frac{Pr[M(A) \in S]}{Pr[M(B) \in S]} \leq e^{\epsilon}$$

In the above introduction to differential privacy, neighboring datasets were defined as datasets with and without the data corresponding to a single individual.

Expanding the above definition off differential privacy, approximate differential privacy (also known as (ϵ, δ)-differential privacy) additionally states that this ”protection guarantees” may fail with a probability of δ, i.e. with a probability of δ anything may happen. The following definition formalizes this notion ([6], definition 2.4):

Definition 2.2 (Approximate Differential Privacy)**.** *A randomized algorithm M with domain $\mathbb{N}^{|X|}$ is (ϵ, δ)-differentially private if for all $S \subseteq Range(M)$ and for all $x, y \in \mathbb{N}^{|X|}$ such that $||x - y||_1 \leq 1$:*

$$Pr[M(x) \in S] \leq e^{\epsilon} \cdot Pr[M(y) \in S] + \delta$$

Note that if δ is set to 0, this definition collapses to Definition 2.1, i.e., differential privacy is just a particular case of approximate differential privacy. As such, we will always refer to this definition in the following, rather than Definition 2.1

While this definition gives weaker guarantees than standard differential privacy, it allows for more algorithms and different compositions. In particular, as will be explained in the following, different types of noise may be used to achieve differential privacy. Some noise types cannot be used for standard differential privacy since they always imply an $\delta > 0$. In addition, some theorems to calculate the differential privacy cost (in terms of ϵ and δ) of multiple applications of algorithms to the same data set are only valid for differential privacy.

Next, a method to achieve differential privacy for some $\epsilon > 0$ is discussed. The basic idea is simple: By adding randomized noise to the output of an algorithm, Definition 2.2 can be achieved for some ϵ. Different values of ϵ can be obtained depending on the kind and magnitude of added noise.

Additive Noise Mechanism

To achieve differential privacy for some function f on a dataset A, some noise needs to be added to $f(A)$. Obviously, the amount of noise must depend on the function, with functions that are more ”sensitive” to small input changes (i.e. where a small input change can lead to a very different result) requiring more noise than other functions. The following definition formalizes this ([6], Definition 3.1):

Definition 2.3 (l_1-sensitivity)**.** *The l_1-sensitivity of a function $f : \mathbb{N}^{|X|} \rightarrow \mathbb{R}^k$ is:*

$$\triangle f = \max_{x,y \in \mathbb{N}^{|X|}, ||x-y||_1 = 1} ||f(x) - f(y)||_1$$

To achieve differential privacy, the Laplace Distribution can be leveraged ([6], Definition 3.2):

Definition 2.4 (Laplace Distribution)**.** *The Laplace Distribution (centered at 0) with scale b is the distribution with probability density function:*

$$Lap(x|b) = \frac{1}{2b} exp\left(-\frac{|x|}{b}\right)$$

The variance of this distribution is $\sigma^2 = 2b^2$. We will sometimes write $Lap(b)$ to denote the Laplace distribution with scale b, and we will sometimes abuse notation and write $Lap(b)$ simply to denote a random variable $X \sim Lap(b)$.

As hinted on earlier, to achieve differential privacy for some function f, we need to add some noise to f’s output. We model this noise as random variables that follow a Laplace distribution ([6], Definition 3.3) to obtain differential privacy:

Definition 2.5 (Laplace Mechanism)**.** *Given any function $f : \mathbb{N}^{|X|} \rightarrow \mathbb{R}^k$, the Laplace mechanism is defined as:*

$$M_L(x, f(.), \epsilon) = f(x) + (Y_1, ..., y_k)$$

where Y_i are i.i.d. random variables drawn from $Lap(\triangle f/\epsilon)$.

Using the Laplace Mechanism as described above guarantees ϵ differential privacy ([6], Theorem 3.6):

Theorem 2.6 (Laplace Mechanism Correctness)**.** *The Laplace mechanism preserves $(\epsilon, 0)$-differential privacy.*

This theorem assures that if by adding proper amounts of noise, any goal in terms of ϵ can be achieved, i.e., any privacy goal can be achieved by paying the price in terms of noise. The amount of noise increases with smaller values for ϵ and decreases with smaller values for $\triangle f$. From this follows that for many aggregation functions (e.g., summing, counting, and averaging), the amount of noise relative to the output size decreases if the aggregation takes more entries into account (assuming a fixed ϵ), i.e., aggregations over larger datasets may provide more accurate results with the same level of privacy (in terms of ϵ)

Composition Theorem

To determine what happens if multiple differentially private algorithms on the same dataset are used is the next logical step. Intuitively, it makes sense that privacy guarantees erode since, with each differentially private result, more and more insight into the dataset can be gained. The most general result is known as sequential composition ([6], Theorem 3.16)

Theorem 2.7 (Sequential Composition)**.** *Let $M_i : \mathbb{N}^{|X|} \to R_i$ be an an (ϵ_i, δ_i) differentially private algorithm for $i \in [k]$. Then if $M_{[k]}(x) = (M_1(x), ..., M_k(x))$, then $M_{[k]}$ is $(\sum_{i=1}^{k} \epsilon_i, \sum_{i=1}^{k} \delta_i)$ differentially private.*

Note that no assumptions other than that each M_i is (approximate) differentially private have been made. This theorem holds in basically any situation involving differential privacy, but more specialized theorems (e.g. [31]) may give better bounds in certain situations. For this thesis, sequential composition and parallel composition ([22], Theorem 4) will be enough:

Theorem 2.8 (Parallel Composition)**.** *Let M_i each provide ϵ-differential privacy. Let D_i be arbitrary disjoint subsets of the input domain D. The sequence of $M_i(X \bigcap D_i)$ provides ϵ-differential privacy.*

The problem with the above definitions and theorems is that they are defined with traditional databases in mind. However, this thesis focuses on unbounded time-series data, which often have markedly different data schemas (see, e.g., Apache Kafka [1]). Towards defining differential privacy for time-series data, first, a general definition of adjacency is needed ([38], Definition 2.4):

Definition 2.9 (X-Adjacent Data Streams)**.** *Data streams (or stream prefixes) S and S' are X-adjacent if they differ only in the presence or absence of any number of occurrences of a single element $x \in X$. In other words, if all occurrences of x are deleted from both streams, then the resulting streams should be identical.*

This then gives rise to the following notions of differential privacy [38]:

Definition 2.10 (Event-Level Differential Privacy)**.** *Here, considered streams are considered as adjacent if they differ in a single event. In other words, X is defined as the set of possible events in Definition 2.9 for two streams consisting of many events. The streams (or stream prefixes) are then considered adjacent if there is a single event, whose deletion from both streams (or stream prefixes) would make the streams identical.*

Definition 2.11 (User-Level Differential Privacy)**.** *On the other hand, in user-level differential privacy, streams are considered adjacent if they differ in the presence or absence of a single user. In other words, X is defined as the set of sets of events which each belong to a single user in Definition 2.9. In other words, streams are user-level adjacent if and only if by deleting from both streams (or stream prefixes) all events belonging to a single user, these two streams (or stream prefixes) would become equal.*

Many different notions of differential privacy on time-series data exist on top of the above ones. For example, w-event-level differential privacy [18], which is a compromise between user-level and event-level differential privacy. However, the above two definitions are sufficient for this thesis.

k-Anonymity

k-Anonymity tries to define the level of privacy of a dataset, regardless of the algorithms executed on this dataset. This idea markedly contrasts with (approximate) differential privacy, which measures the privacy impact of certain algorithms in terms of ϵ and δ. The idea is that in a k-anonymous database, for any entry in this database, at least $k - 1$ entries are present in the database, which have the same value for any attribute that may serve as an identifier [33]:

Definition 2.12 (k-Anonymity)**.** *Each release of data must be such that every combination of values of quasi-identifiers can be indistinctly matched to at least k individuals.*

As an example, consider a database that contains data from customers of some business. If this database is k-anonymous, even if, e.g., the customer's address, age, and name are known, there should always be at least k entries that match these attributes. In this example, $k = 1$ seems to be the only plausible value due to the combination of name, address, and age likely being unique. Towards achieving higher values of k, a possibility would be only to save the first letter of a person's first name, assign each person to one of five age groups and assign each person to a more extensive area instead of an accurate address. After such modifications, it seems plausible that a big part of the dataset conforms to k-anonymity for greater values than $k = 1$. Simply removing non-conforming entries would extend this property to the whole remaining dataset.

In this thesis, we will not consider k-Anonymity further and instead focus on differential privacy. As can be seen from the above definition and example, k-Anonymity is again defined for traditional databases. Therefore, this definition is not directly applicable to infinite time-series, which is the focus of this thesis.

In addition, applying k-Anonymity to achieve privacy is not that straightforward in all cases (see, e.g., [35]) and does not give guarantees comparable to differential privacy. Indeed, according to Narayanan and Shmatikov, for some kinds of databases, k-anonymity does not provide any meaningful privacy guarantees [24]. To the best of our knowledge, such limitations do not apply to differential privacy, which gives robust guarantees without any additional assumptions.

2.2.2 Privacy Transformations

Privacy transformations relate to the concept of modifying data such that less information is contained to protect the privacy of data subjects. In this section, common privacy transformations used in practice are covered, and the provided privacy protection guarantees are discussed. Most transformations considered here are also used in privacy platforms such as Zeph [4] or Privitar [27].

Table 1 provides an overview over the considered privacy transformations.

	PRIVACY TRANSFORMATION	SHORT DESCRIPTION
DATA MASKING	Tokenization	Replace sensitive data with a unique token
	Pseudonymization	Remove any (pseudo-)identifiers from data
	Perturbation	Adding noise to data values or outputs of algorithms
	Redaction	Remove data or certain aspects of it
	Shifting	Modify each data subject's entries using a (per subject) fixed formula
DATA GENERALIZATION	Bucketing	First define several buckets and then only store or report in which bucket a value belongs
	Time Resolution	Aggregating data of a data subject during a certain time span
	Population Resolution	Aggregating data of different data subjects

Table 1: An overview of the different privacy transformations

Each of these privacy transformations has its uses, and none of them are appropriate in all situations. To transform data before processing to limit exposure of sensitive data while also still enabling value generation from transformed data may require different privacy transformations with different parameters depending on, e.g., application and data type. As an example, consider the step counting function many modern smartphones support. Health insurance companies may offer discounts if a certain number of steps are reached each day and therefore need access to daily aggregates. On the other hand, statistical analyses of the average number of steps of a country's population only need access to data aggregated over extensive groups of people.

Data Masking Data masking obfuscates sensitive parts of the data while still retaining some usefulness [4]. In the following, privacy transformations that fall under this category from Table 1 are introduced in more detail.

Definition 2.13 (Tokenization)**.** *Tokenization refers to the process of "replacing sensitive data with a unique token" [27]. This token should retain usefulness for some intended purposes while not revealing more information than needed.*

Depending on what the goal is, this process may be reversible or (practically) irreversible. In the first case, tokenization is a form of encryption. It is important to note here that reversible tokenization does not mean reversible by anyone accessing the data. Instead, it usually requires some secret key or access to a database that contains a mapping from tokens to original values. For example, many commercial systems will assign users an id, which can only be mapped to a natural person by a subset of employees with access to a database table that maps this id to, e.g., a name or an address. Nonreversible tokenization is widely used as well, especially in the form of hashes. For example, in payment systems, hashes can be used to verify that the sender owns a secret key without actually revealing the secret key in any message since these messages may be transmitted through insecure channels.

Tokenization can also be deterministic or nondeterministic. In the deterministic case, the same input will always produce the same output. Determinism is, for example, an essential requirement in traditional databases where data is combined across different tables by joining key attributes. On the other hand, nondeterministic means that the same input may (pseudo-)randomly result in different outputs. This process makes linking as before impossible but can help preserve privacy for precisely this reason. Ergo, any sensible form of encryption is also nondeterministic (if one does not have access to the secret key).

Definition 2.14 (Pseudonymization)**.** *Pseudonymization refers to the idea of removing any attributes from data that could lead to an identification of data subjects.*

Essential attributes are fundamentally the same as for tokenization (deterministic vs. nondeterministic, reversible vs. nonreversible). However, these choices have additional consequences in this context. Consider a situation where any data that may lead to identification is tokenized irreversibly. By definition, it is no longer possible to identify the person, which means the data is pseudonymized. On the other hand, if the tokenization is reversible, the data can only be described as pseudonymized as long as this reversal has not occurred or cannot occur. Suitable methods include storing personal data separately and only accessible for authorized persons.

The impact of deterministic vs. nondeterministic is again mostly in how data can be linked. This choice may impact what information needs to be tokenized. For example, neither age nor a ZIP code will typically be enough to identify a person (assuming several people correspond to any given ZIP code or age). However, if these attributes can be linked, it may be possible to identify a person (see also Definition 2.12: k-Anonymity in this context).

Definition 2.15 (Perturbation)**.** *Perturbation relates to the notion of adding noise to original data values or outputs of algorithms, e.g., to achieve differential privacy (Definition 2.2).*

As explored further and more formally in Definition 2.2, the addition of more noise usually corresponds to better protection of privacy of data subjects but at the cost of reducing the query accuracy and utility of data. However, what kinds of guarantees are given (if any) by added noise depends on the type of noise. As stated in Theorem 2.6, noise generated from a Laplace Distribution can be used to achieve $(\epsilon, 0)$ differential privacy for any $\epsilon > 0$, while gaussian noise provides approximate differential privacy.

However, perturbation does not need to be with differential privacy in mind. Noise can also be added, e.g., from some bounded uniform distribution to data values or algorithmic results without a specific privacy model in mind. In this case, there are no formal guarantees, unlike when using differential privacy. However, such a transformation may still be of some use for privacy protection or other purposes, like preventing overfitting a machine learning model to its training data.

Definition 2.16 (Redaction)**.** *Redaction refers to omitting some data or aspects of it.*

For this thesis three types of redaction are essential:

The first type refers to simply omitting specific fields/attributes in a dataset. For example, consider a customer of some health service who regularly measures their blood pressure and heart rate. For statistical analyses, the customer may grant this service access to the heart rate but omit the blood pressure. On the other hand, a doctor may access the data without restriction to decide on optimal treatment.

The second type of redaction refers to only omitting values if they fall into a particular range. For example, heart rates in a ”normal” range could be omitted, while values that fall outside this range are still transmitted. Alternatively, data may not be transmitted unless the heart rate goes into a critical region way above ”normal” ranges, in which case emergency responders can automatically be notified.

Finally, the third type of redaction which is of interest in the context of this thesis, is to redact specific statistics of a dataset (e.g., the variance of a value) while giving access to others (e.g., mean). Of course, this is only meaningful if no access is granted to the underlying data directly.

Especially the first type of redaction gives the best privacy guarantees possible for the data subject regarding the omitted data. However, on the flip side, no value can be generated from omitted data. As discussed in Section 2.1, more modern definitions of privacy often also consider the effects of not sharing data on other actors than the data subject (e.g., an organization or society at large). Such definitions explain why the other types of redaction are also of great importance in a privacy-focused system.

Definition 2.17 (Shifting)**.** *In shifting, the idea is that each data subject chooses a random offset and then only transmits data with this offset added.*

In contrast to Definition 2.15: Perturbation, this preserves relationships such as $<, >$, and $=$. for data of a single user, which may be necessary to preserve the utility of specific data. Due to this reason, shifting is commonly applied to dates and times due to it preserving the order of events for a single user. Even with shifted dates, the amount of time between events/entries and their ordering is preserved, which is the primary concern in many applications and not the exact date itself. However, this transformation still has the drawback of not allowing to link this information to other date-dependent information, such as weather information on a particular day or other user’s data. Of course, this technique could also be applied to other data, e.g., weight or height, to track relative changes without leaking actual values.

While the more excellent utility is advantageous in many cases, this transformation provides fewer guarantees than per-value random perturbation. If the actual value of any single shifted data value is known, the offset chosen by a particular data subject can be calculated. From this, the actual value of all shifted data can be derived.

In summary, this transformation does offer little privacy protection for data subjects. However, more analyses are possible in general, compared to more disruptive transformations such as perturbation in combination with differential privacy. As with any other transformation, shifting offers a specific tradeoff that may or may not be appropriate for a given application, purpose, and setting.

Data Generalization Data generalization describes techniques reducing data fidelity [4]. In other words, the granularity and, therefore, the informational value of data is in some way reduced, e.g., by only releasing aggregate data or shrinking the size of the data domain. Below, the three most common types of data generalization are discussed. Since they are orthogonal to each other, they can be combined freely.

Definition 2.18 (Bucketing)**.** *This transformation describes a technique in which, instead of giving access to raw values, several buckets are defined, and it is only reported in which bucket each value falls. In other words, the domain of values is reduced to a smaller set of buckets.*

An important example of this transformation is its application to location data. For example, sharing one's address has a distinctly different impact on privacy than merely sharing one's country of residence. A similar effect could also be reached by using Perturbation (e.g., on coordinates), but a bucketing approach can take into account specific circumstances like country borders or population density, which is not possible with standard Perturbation. Bucketing location data is a natural transformation for many use cases. A user may not want to share its exact location with a service but may not mind reporting in which area they currently are. This transformation enables the service to gain essential data for decision-making while respecting the user's privacy concerns. By appropriately designing the buckets, favorable tradeoffs between privacy and extracting valuable information from data can be made.

Definition 2.19 (Time Resolution)**.** *This refers to lowering the temporal resolution of the data by aggregating data together that lies within a specific time frame.*

Consider the example of an intelligent car transmitting to the car manufacturer how far it has moved since the last update. If updates happen every minute, the car manufacturer would see exactly when and when not the car is used, with which many customers might not be comfortable. On the other hand, sharing the total distance only at the end of each month still enables elemental analyses for the manufacturer, like how far people usually drive during the first year, while respecting users' privacy concerns.

Definition 2.20 (Population Resolution)**.** *This refers to only releasing data that is the result of an aggregation over a certain amount of people.*

Intuitively, the idea is to "hide" individuals' data in a group, such that private data is not exposed. As an example, consider feedback ratings for some services. Such feedback is usually made available only in aggregate form, e.g., it might only show average and variance for each asked metric, but not more. This information is enough to determine whether or not many or a substantial part of customers are satisfied or not while not exposing any individual's responses.

Aggregate Functions

For both aggregation over time and population (Definition 2.19 and Definition 2.20), what kinds of aggregation can take place has not been discussed yet. For example, a simple statistical query (e.g., maximum, mean, or variance of a value) might need to be executed. Alternatively, the data may be supposed to be used in more advanced ways (e.g., to train trend detection models).

In general, these aggregate functions can be applied both with aggregation over time and aggregation over population. However, greater care should be taken if such functions are used by a randomized algorithm that tries to reach certain differential privacy goals (i.e., in terms of ϵ and δ). For the Laplace mechanism (Definition 2.5), the l_1-sensitivity (Definition 2.3) of the aggregate function determines how much noise needs to be added to reach certain values of ϵ. Different mechanisms to reach differential privacy do exist but still depend on which function should be made differentially private.

3 Related Work

This thesis presents a system design that manages and handles heterogeneous privacy policies integrating state-of-the-art privacy transformations by optimally using non-replenishable privacy resources to fulfill a set of queries defined by the service provider. In this section, we discuss work that is of relevance to this line of work.

3.1 Privacy Policy Enforcement

One key aspect of this thesis is defining and automatically enforcing a user-centric privacy model. This section explores prior work that focuses on policy enforcement, though without the same user-centric approach, i.e., with the focus on organization-wide privacy policies and enforcing role-based access controls. To this end, we introduce four systems that are relevant in this regard:

- *Towards Muliverse Databases [21]:* The focus of this work is providing an extension to SQL database systems that allows setting policies such that each user only sees a policy-compliant view (”universe”) of the database [21]. Similar to our thesis, this separates policy management from other application logic by automatically enforcing these policies and creating policy-compliant views. However, the authors consider traditional databases and role-based access controls and do not follow a user-centric approach. We instead focus on streaming data, more commonly associated with platforms such as Apache Kafka [1] instead of systems using relational models, and focus on state-of-the-art privacy transformations which apply to any access regardless of role.
- *Privacy Integrated Queries (PINQ) [22]:* In PINQ, the author presents a platform to analyze data from traditional databases by providing a wrapper that checks and enforces certain privacy guarantees. Similar to this work, the focus is on differential privacy due to its robust guarantees. Privacy Integrated Data Stream Queries [38] extends PINQ by providing automatic enforcement of differential privacy (both on a user- and event-level) on data streams, as is the case in this thesis. However, we go a step further and allow users to have heterogeneous privacy policies that allow setting privacy transformations without differential privacy. We then focus on providing optimal usage of non-replenishable privacy resources given a set of queries and these heterogeneous privacy policies.
- *Privitar [27]:* Privitar is a platform that allows setting policies that are valid across an organization. Privitar then automatically enforces these policies

by creating distinct ”protected data domains” for different purposes, i.e., policy-compliant views. Such policies consist of a set of rules, where each rule is a combination of techniques, similar to what we refer to as ”transformation chains” in Section 4.2.2. Unlike our thesis, however, Privitar focuses on compliance and enforcing company-wide privacy policies. We instead have chosen a user-centric approach and handling a diverse set of privacy policies that result from our user-centric approach.

- *Zelkova [17]:* Zelkova is a system that analyzes policies for Amazon's AWS. It can automatically derive specific implications of policies in place. For example, Zelkova is used to derive if the applicable set of policies leads to particular resources being publicly accessible. It decides such questions by comparing the policies in place with a policy corresponding to a particular question (e.g., if a resource is publicly available) using an SMT solver. Zelkova and related research are orthogonal to this thesis and can help understand what implications a particular privacy policy has, which may help users and experts make informed decisions when setting their privacy policy.

3.1.1 User-centric

More closely related to our thesis, some prior work also considers a user-centric approach to privacy, in which different policies can be associated with different users. We consider here the three most important approaches in connection with this thesis:

- *Zeph [4]:* Zeph offers a platform where users can restrict access to their data and enforce these restrictions cryptographically. Data producers will only transmit encrypted results, while a privacy controller must authorize any decryption. This privacy controller will only provide the necessary decryption information if queries conform to users' privacy policies. A component called the ”policy manager” is responsible on the provider side to match users and queries to make sure no privacy policy is violated, which corresponds to our privacy planner. While the authors consider a greedy, online approach for allocating non-replenishable privacy resources, we consider an offline, batch-oriented approach for our privacy planner, making optimal use of these limited resources. This thesis is complementary to Zeph and could be used to improve upon the system presented there.

- *Ancile [3]:* Ancile allows users to specify use-based privacy controls, which focus on limiting how data can be used. The authors present a modified version of Avenance, a language designed for use-based privacy, to specify such restrictions as regular expressions. Instead of why the data is being accessed, we instead focus on giving users control over what kind of inferences can be made from their data. While we do not support data-dependent policies, we support a much richer set of privacy transformations and specifically consider constraints that arise from them (most notably regarding the reusability of data, which is expressed in a privacy budget). To appropriately support restrictions arising from this budget, we present our privacy planner. In Ancile, such a component is not present since different constraints are taken into account.

- *Riverbed [36]:* In Riverbed, each user can specify a privacy policy for each web service they use. Similar to this thesis, the authors imagine a system where experts define appropriate privacy policies. This policy specifies, most importantly, which web services are allowed to handle the users' data, what these services' software stacks are allowed to look like, and some restrictions on how data can be handled (e.g., whether data may be aggregated). Therefore, these restrictions are very different from our policies: We provide a rich set of privacy transformations. Users can limit what kind of inferences can be made but are agnostic regarding the software stack used to obtain such a privacy-compliant view.

3.2 Privacy Accounting

A key difference to other related work so far was that we consider a rich set of state-of-the-art privacy transformations, which together introduce a substantial number of constraints. Most importantly, we consider privacy budgets, limiting the amount and type of transformations that can be run on data. Compared to binary role-based access restrictions, such policies can provide better tradeoffs regarding data utility and protect users' privacy. In the following, we will present two critical systems that prominently focus on keeping track of and optimizing the allocation of privacy budgets.

- *Sage [19]:* Sage is a system for training machine learning models on a continually growing set of data. They address the tradeoffs between model accuracy and differential privacy budget consumption. To this end, they subdivide the stream of incoming data into blocks, each with a certain differential privacy budget. Through this, they can enforce event-level differential privacy: If a blocks differential privacy budget is used up, this block is retired. While we also support event-level differential privacy in this thesis, we also support a rich set of privacy transformations, including transformations without differential privacy and user-level differential privacy. In addition, we focus on dealing with a large set of potentially different privacy policies. At the same time, sage only considers a single privacy policy (a budget in terms of ϵ and δ for differential privacy).

- *Scheduling Privacy Budget:* This work presents a scheduler (DPF) to enforce a global differential privacy guarantee, given multiple batch workloads which consume specific amounts of differential privacy (in terms of ϵ and δ) [20]. To this end, data streams are subdivided into blocks whose form depends on whether user-level, event-level, or a compromise between the two differential privacy notions should be supported. These options go further than Sage [19], which only supports event-level differential privacy. They show that the DPF algorithm outperforms traditional scheduling algorithms for privacy resources since standard approaches assume that all resources are replenishable (e.g., CPU cycles). The authors then describe their implementation in Kubernetes. While they consider an online setting and consider a greedy approach, we instead consider an offline, batch-oriented approach to distribute privacy resources. Through this, we can come up with optimal solutions given providers' preferences. In addition, we allow a heterogeneous set of privacy policies that also allows other restrictions than differential privacy. Finally, they optimize for fairness between different workloads. On the other hand, we introduce the concept of profits such that the provider can express preferences regarding which queries should be prioritized.

4 Design

This section we start with brief overview of the system, then we discuss the API available for both users and the service providers to specify different privacy policies. Finally, we detail the design of the privacy planner. We start by presenting the query API and which problems and constraints arise while fulfilling queries under a heterogeneous set of privacy policies. Then, we present two approaches for the privacy planner: The heuristic approach and the ILP approach.

4.1 System Overview

As more privacy solutions emerge and systems start to adapt more fine-grained privacy controls, the new fundamental issue of privacy management arises. Privacy management is prevalent across almost all privacy-preserving systems that go beyond a specialized, ad-hoc privacy-preserving data collection mechanism [5, 8]. The consequence of giving more privacy controls to the user, i.e., a user-centric model for privacy, is that a service provider must handle data with heterogeneous privacy policies. The heterogeneous set of privacy restrictions leads to a logical partitioning of the data and the underlying privacy restrictions dictate when and under what circumstances it is possible to combine data from different partitions. Observe that the problem of data with multiple privacy policies goes beyond systems that follow a user-centric model for privacy. Services that operate across regions where different legal privacy restrictions apply also end up partitioning the data.

A naive solution can be realized by treating individual partitions as data silos and running only compatible queries within a partition but never across. This however severely harms the utility that could be extracted from data. Moreover, even with this simple solution, it remains unclear how to determine that a query is compatible with certain privacy restrictions. In this thesis, we want to go a step beyond with a new system design. First, the system design provides the infrastructure to determine whether a particular query is compatible with the privacy restrictions of a partition. Second, the design also allows combining data from different partitions.

In addition to the challenge of data-partitioning, privacy management also needs to address the challenge of managing privacy resources. Privacy is a finite, non-replenishable resource and privacy restrictions impose limitations on using the same raw data in multiple queries. As a result, different queries compete for the same data, and privacy management needs to orchestrate this competition. For example, data used in an aggregation query (e.g., sum over 1000 users) cannot be used elsewhere to prevent differencing attacks (a type of reconstruction attack).

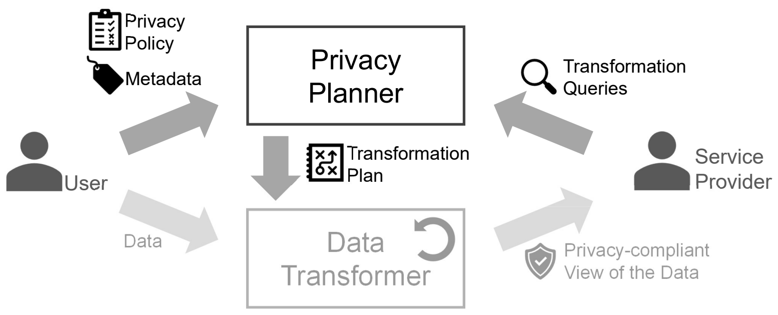


Figure 4.1: An overview over the different components of a end-to-end privacy solution, with the parts which are the focus of this thesis in black.

Alternatively, in differential privacy, a composition theorem controls the privacy cost for running multiple queries.

In this thesis, we consider privacy management for a system that follows a user-centric privacy model. Figure 4.1 gives an overview of the components. Users of a service define a privacy policy that outlines what privacy restrictions apply to their data. Apart from the privacy policy, a user also provides additional metadata that provides further information about the data (e.g., the users' age or country of residence), and finally, a user submits streams of raw (sensitive) data.

We follow the approach from Zeph and Privitar [4, 27] and leverage privacy transformations to bring incoming raw data into the form of a privacy-compliant view. The privacy-compliant view respects all user-defined privacy restrictions, and the view is available for arbitrary post-processing in existing data processing systems.

The privacy planner of the service provider is responsible for the privacy management. It collects privacy policies and metadata of all available data streams, tracks the consumption of privacy resources from previous queries, and provides an interface for expressing transformation queries. The service provider uses the query API to specify transformations that allow extracting utility from the data while respecting a user's privacy policy. Based on the privacy policies, metadata, remaining privacy resources, and the set of transformation queries, the privacy planner constructs a transformation plan which specifies the privacy transformations to execute and on top of which parts of the pool of available data. In essence, the transformation plan is an assignment or matching between the available data and the transformation queries.

Finally, a data transformer receives the transformation plan and executes the privacy transformations to construct a privacy-compliant view of the data.

This thesis focuses on privacy management, which involves expressing privacy policies and transformation queries and the privacy planner to construct a transformation plan. The data transformer which uses the transformation plan to construct privacy-compliant views is outside the scope of the thesis.

4.2 User-centric Privacy Design

Our design follows a user-centric model for privacy, i.e., allows users to express their preferences on how their data should be handled. These privacy policies are then one of the inputs for the privacy planner that ensures that queries are matched with data according to users' privacy policy. This section gives insight into the parts of our system concerned with providing users control over how privacy preferences are assigned and applied to their data.

Most existing services generally use a privacy policy in the form of a long legal document. We could borrow this idea also for a system that follows a user-centric model for privacy. In such a strawman, users express their privacy policy in the form of an unstructured text document. Service providers would collect privacy policies and analyze them to extract the privacy restrictions of the data. However, this strawman has two issues that render the approach infeasible. First, writing legal documents such as privacy policies requires a legal background and training. Second, even if we assume that users can express written privacy policies, it would be almost impossible for a service provider to determine how to use the data. Indeed, this approach does not scale for services with a large number of users. As a result, it is evident that in a user-centric model, we require structured privacy policies in a machine-readable form while keeping the complexity for end-users low.

In the model of this thesis, a privacy policy describes what type of privacy transformations a service provider needs to perform before being authorized to use the data. The general approach is structured similarly to Zeph [4]. However, in this thesis, we make it more expressive with additional functionalities to consider the challenges of the privacy planner better. In Figure 4.2, we provide an overview of the system design that allows users to express their privacy preferences.

What type of privacy transformations are sensible depends on the underlying data. As a result, it is natural to build privacy controls on top of existing data schemas. Every application working with streaming data already requires a *data schema*. The only proposed conceptual difference is adding some additional attributes to the schema to enable privacy controls.

A challenge for data privacy is to identify transformations that respect users' privacy preferences and, at the same time, provide utility for the service provider,

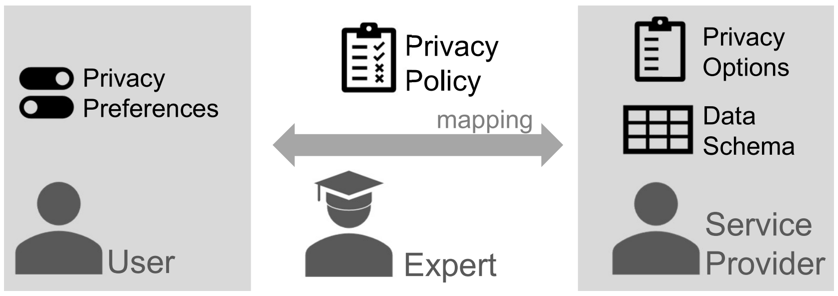
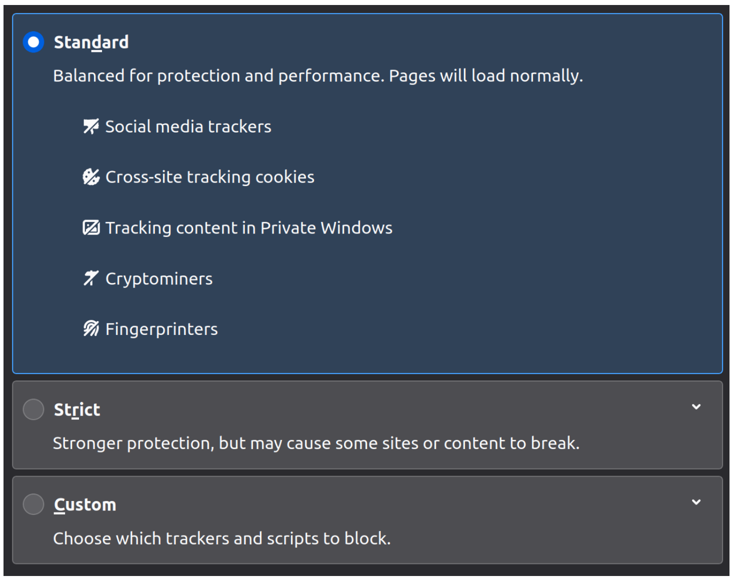


Figure 4.2: A closer look at how users can be assigned privacy policies in practice.

i.e., such that the service provider can offer the desired functionality. We address this challenge with the idea that the service provider defines a selection of *privacy options*. A single privacy option is a sequence of privacy transformations (called ”chain”) with appropriate parameters. In essence, users create privacy policies by selecting the acceptable options for their data, or in other words, a *privacy policy* is a (many-to-many) mapping of privacy options to different parts of data.

While our system design enables users to explicitly set their privacy policy, doing so for each application can quickly overwhelm users and may require expert knowledge to understand all implications. Complex and time-consuming privacy policies are not a new phenomena [39, 34] . In that regard, our model is no different and arguably even worse because our privacy policies are more precise and verbose. To solve this issue and make the system also accessible to non-expert users, we envision that users only set their high-level privacy preferences, which are valid across applications. Afterward, for each application, privacy experts (e.g., from the EFF [7]) design appropriate privacy policies that correspond to the high-level privacy preferences. Selecting a reasonable choice of privacy preferences is outside the scope of this thesis. However, controlling these preferences could be designed similar to how tracking preferences are set in the Firefox browser, see Figure 4.3. Note that advanced users are still able to set their customized privacy policies.

In Section 2.2, we presented existing privacy transformations that are used in practice and discussed different privacy models. Before we dive into augmenting data schemas with privacy options and expressing privacy policies, we revisit the transformations and privacy models and highlight what we support. We conclude the user-centric privacy design with a case study that shows how the different parts look in a concrete application.


[● Standard]
Balanced for protection and performance. Pages will load normally.

Social media trackers

Cross-site tracking cookies

Tracking content in Private Windows

Cryptominers

Fingerprinters

Strict
Stronger protection, but may cause some sites or content to break.

Custom
Choose which trackers and scripts to block.

Figure 4.3: Different options in firefox regarding tracking preferences.

4.2.1 Supported Privacy Models and Transformations

As a prerequisite for expressing privacy policies, we first define what kinds of privacy models we can support and which privacy transformations using which parameters we take into account. In Section 2.2 we list a general description of privacy models and transformations. This section specifically focuses on the privacy models and transformations that are part of our system, together with the parameters we make available to modify transformations' properties.

Supported Privacy Models In Section 2.2, two privacy models were introduced: k-Anonymity and (approximate) differential privacy (DP). In our system, we only make (approximate) differential privacy available for two main reasons: First, differential privacy is the gold standard of privacy protection, and for many types of algorithms, modifications to include differential privacy already exist. In principle, such algorithms can then be used in conjunction with our privacy planner. Second, differential privacy can easily be applied to time series data, which is the focus of this thesis. Finally, substantial prior work exploring the privacy guarantees extended by differential privacy on time series data, depending on how the definition is applied, already exists.

Specifically, our systems supports (ϵ, δ) approximate differential privacy (Definition 2.2) in the form of event-level differential privacy (Definition 2.10) and user-level differential privacy (Definition 2.11). We support both sequential composition and parallel composition, which are automatically applied depending on the data schema (what we call ”privacy groups” in later sections. To achieve differential privacy, we support adding different types of noise, including Laplacian noise (Definition 2.4).

However, we also support various privacy transformations without formal guarantees. For specific applications, such transformations are an invaluable tool, as some kinds of analyses can respond very poorly to the kinds of noise added for differential privacy. For example, consider a sequence of events, each with a timestamp. If we add random noise to this timestamp, events might not be in the same order anymore, leading to very misleading results. At the cost of potentially less privacy protection, we could instead choose a random offset only per user instead of per event (which corresponds to a privacy transformation called ”shifting”), which still enables such analyses

Supported Privacy Transformations. In Section 2.2, a variety of transformations that can be used to enhance users' privacy were introduced. Here, we detail each transformation available in our system and which parameters can modify its behavior.

- Field Redaction: *RedField*

  The most effective way to respect users' privacy: Simply not transmitting or deleting particular data streams. This transformation corresponds to the first type of redaction as explained in Definition 2.16.

- Range Redaction: *RangeRed*

  This transformation corresponds to the second type of redaction from Definition 2.16. If this transformation is applied to a data stream, only values outside of a certain range are transmitted. We therefore have the following parameters:

  - min: Defines the minimum of the redacted range (included).
  - max: Defines the maximum of the redacted range (not included).

- Tokenization: *Tokenize*

  This transformation transforms each value into a token, which keeps only certain properties of the original data. Tokenization is further described in Definition 2.13. Since tokenization functions usually need to be adapted to specific underlying data and use cases, providers can specify custom tokenization functions as an argument. These functions may or may not be reversible or randomized.

  - $func$: A function that is applied to every value that is to be tokenized.

- Shifting: *Shift*

  Shifting describes each user only transmitting data after adding a, per user, fixed offset, as defined in Definition 2.17. The offset is chosen from a uniform distribution:

  - $size$: defines that the offset is chosen uniformly at random from the range $(-size/2, size/2)$, where $-size/2$ and $size/2$ are not included.

- Perturbation: *Pert*

  Similar to shifting, all values are shifted by a certain random offset. However, as defined in Definition 2.15, this offset is not fixed per user, i.e., is chosen anew for each value. Again, the offset is chosen from a uniform distribution, which means we have the following parameters:

  - $size$: defines that the offset is chosen uniformly at random from the range $(-size/2, size/2)$, where $-size/2$ and $size/2$ are not included.

- Differential Privacy Perturbation: *PertDP*

  This transformation is also just a perturbation as defined in Definition 2.15, with the difference being that only types of noises that can be used towards achieving differential privacy are available. Following this, the parameters specify which ϵ and δ should be achieved using the specified types of noise. These parameters, together with some information about the range of values in a specific data stream and which notion of differential privacy (user-level or event-level) should be achieved, then implies the parameters used for generating the noise.

  - $DP\text{-}notion$: event-level or user-level
  - ϵ: Which value of ϵ for differential privacy we want to achieve.
  - δ: Which value of δ for differential privacy we want to achieve.

  - $noiseType$: What type of noise is used, e.g. gaussian or laplacian. Not all types of noise can be used to achieve all combinations of ϵ and δ.

- Bucketing: *Bucket*

  As described further in Definition 2.18, bucketing is a transformation where each value is assigned to a certain bucket. The released data then only contains which bucket a value is a part of and not the original value, thereby reducing the amount of information. For ease of management, each bucket can also be assigned a label:

  - $buckets$: A list of individual bucket definitions. Each such definition contains two numbers min and max, which means that all values in the range (min, max) should be assigned to this bucket (including min, not including max). Optionally, a $label$ can be used to name a bucket. For multidimensional data types (e.g., location coordinates), multidimensional buckets can be set.

- Time Resolution: *TimeRes*

  The idea here is to define a time range (e.g., a day) and only release data aggregated over this range of time, as defined in Definition 2.19. Which exact statistics are released can also be set, which implies that this transformation also corresponds to the third type of redaction from Definition 2.16.

  - $nSeconds$: How many seconds the range of time that data should be aggregated over has.
  - $AggrFuncs$: Exactly which aggregate statistics are released (e.g., count, sum, variance).

- Population Resolution: *PopRes*

  The concept is the same as for time resolution, but the aggregation is over people instead of time (as defined in Definition 2.20). Again, which statistics are released is exposed as an argument to this transformation.

  - $nPeople$: How many people should be aggregated over.
  - $AggrFuncs$: Exactly which aggregate statistics are released (e.g. count, sum, variance...).

- Unmodified Data Release: *UnmodRelease*

  Finally, this ”transformation” corresponds to not doing any privacy transformation, i.e., just releasing the data to the provider without restriction.

With this, we conclude our overview of the supported privacy models and transformations. The goal was to capture the most significant transformations in use today or commonly used in recent research, so many different privacy preferences users can be expressed using these transformations. However, it is worth noting that our approach is general enough to handle many other privacy transformations with minimal modifications.

4.2.2 Expressing Policies

In this section, we present data schemas and privacy options in detail (also see Figure 4.2), and how these parts come together to form a comprehensive privacy policy.

Data Schema. Towards creating suitable privacy policies for an application, it is essential to know precisely which data an application is processing. As we focus on time series (or streaming) data, data schemas usually already exist, e.g., for Apache Kafka [1] or for some SQL database. Our data schema expands such schemas with some specific information that helps towards achieving favorable tradeoffs. As with existing schemas, the service provider defines our extended schemas that include privacy controls.

Our data schema is built around three core concepts: event streams, value streams, and privacy groups.

- A value stream is a continuous flow of values, each of which has an associated timestamp. For example, location data transmitted every few seconds to a navigation application would be classified as a value stream.
- An event stream is a collection of value streams that share timestamps. For example, if we have a sensor that measures air pressure and wind direction once a second, value streams for air pressure and wind direction can be unified into a value stream.
- Finally, privacy groups are also collections of value streams but orthogonal to event streams. Instead of signifying common timestamps, these are used if values are closely correlated or imply each other. For example, consider location data and data on the movement speed and direction of a person. These two types of data each imply the other (modulo the starting position). By designating such value streams as being in the same privacy group, our system can respect such impacts on privacy.

Following this discussion, we define the schema for a value stream as follows:

- **name** (required)

  The name of this value stream.

- **type** (required)

  The data type of this value stream. The type may be any numerical type, either as a scalar value or a tuple (e.g., coordinates to represent location).

- **privacy group** (optional)

  Defines to which privacy group this value stream belongs. If not defined, this value stream forms a privacy group with itself as the only member.

- **range** (optional)

  For each scalar value in the data type (i.e. once for each entry of a tuple), define the range of available values as (min, max) (min included, max excluded). If not defined, we assume the full range the chosen data type can represent.

- **description** (optional)

  Optional description

An event stream is then simply a list of value streams as defined above.

While range is an optional attribute, it is of central importance if differential privacy is to be used. In which range values need to lie directly gives an upper bound for the l_1-sensitivity (see Definition 2.3) of many types of aggregations. In turn, this upper bound (together with the parameters specified for PertDP) determines the amount of noise added to the results. Unsurprisingly, a smaller range implies less noise needs to be added, i.e., results conforming to differential privacy will be more accurate for fixed values of ϵ and δ

Example 4.1 provides a straightforward example of how a value stream could be instantiated. A more comprehensive example involving multiple value streams can be found in Section 4.2.2.

Example 4.1 (Value Stream)**.**
name*: "heartRate"*
type*:* $long$
privacyGroup*:* 1
range*:* $[0, 250]$

Metadata As already outlined in Section 4.1, metadata will be a crucial component for our privacy planner. Metadata allows providers to filter for specific users in their queries to do more meaningful analyses. For example, querying for film preferences of people under 30 requires that users' have assigned age metadata. We think of metadata as just a value stream but, additionally, assume that there are only infrequent updates (e.g., once a year for age).

In addition to ”core” data, metadata is needed for many types of queries to determine if a specific stream should be included. In this thesis, we consider metadata as just another data stream, albeit with infrequent updates. Typical examples of metadata include age, which changes once a year, or the municipality of residence, which likely changes not more than once every few years.

Most attributes are the same as for normal value streams. An additional boolean-valued attribute **metadata** distinguishes metadata streams from normal value streams. We also make an additional attribute **symbols** available for metadata stream, which is required if the type is set to $enum$ (which is not allowed for regular value streams). The reason for this is that many privacy transformations only work for numeric types. However, by applying bucketing to numeric types (or just encoding enums as integers), the same result can be achieved. This argument expects a list of strings, which defines the list of available symbols. Example 4.2 illustrates this.

Example 4.2 (Metadata Stream)**.**
***name**: ”residentialRegion”*
***metadata**:* $true$
***type**:* $enum$
***symbols**:* [*”North”, ”East”, ”South”, ”West”*]

Metadata streams can either be on a user level or part of event streams like standard value streams. Defining metadata streams as part of event streams indicates that a particular metadata attribute is a property of an event stream (e.g., the type of sensor producing the data for a specific event stream). However, the two options do not have any functional differences.

Transformation Chains After discussing which privacy transformations are available in our system and how to express data schemas, we now introduce transformation chains. Transformation chains are, in essence, a sequence of transformations. By allowing privacy transformations to be combined, additional trade-offs regarding privacy and utility are possible. For example, we may want to specify that data is only released as daily aggregation over ten people, a sequence of two transformations: Aggregation over time and aggregation over people.

At the beginning of Section 4.2, we have defined which privacy transformations are available in our system. Here, we define which combinations of these transformations we support.

While there are almost limitless possibilities for combining transformations, not all combinations make sense. We identify five chains, which include optional steps, to realize a large class of transformations. In our system, we then support any instantiated chain (sequence of transformations including instantiated arguments) which adheres to this model.

Definition 4.3 (Available Chains)**.**

- *Private Sharing Chain: (RangeRed) $\rightarrow$ (Bucket) $\rightarrow$ (TimeRes) $\rightarrow$ (PopRes) $\rightarrow$ (Pert/PertDP)*
- *Tokenization Chain: Tokenize*
- *Deletion Chain: RedField*
- *Metadata Chain: Shift/Bucket*
- *Public Chain: UnmodRelease*

where the following notation was used:

- "()": optional
- "/": one or the other
- "$\rightarrow$": one after the other

So, as an example, bucketing data and then adding noise to the bucket counts would be an instance of the private sharing chain, while first adding noise to data and then bucketing it would not be supported.

Privacy Options Having defined chains, we can now define privacy options, which most notably include an instantiation of a particular chain that defines which combinations of transformations can be done on a particular value stream. For example, a privacy option may specify a chain that mandates bucketing data and then aggregating it over 100 users. To comply with this option, transformations done on data associated with this option need to include these two transformations. In other words, the idea is that a privacy option defines a minimum requirement regarding which privacy transformations are applied to a specific value stream. For a sequence of transformations to be compliant with a privacy option, data must undergo at least the listed transformations (more on this later).

To this end, we define that privacy options are expressed with the following schema:

Definition 4.4 (Privacy Option)**.**

- ***name*** *(required)*
  The name of this privacy option.
- ***chain*** *(required)*
  Which chain from Definition 4.3 this instance belongs to.
- ***transformations*** *(required)*
  The transformations and associated parameters (as defined in the beginning of Section 4.2).
- ***appliesTo*** *(optional)*
  To which value streams can this option be applied. A unique option, metadata, is used to refer to all metadata streams. By default, an option can be applied to all value streams (and metadata streams for the metadata and deletion chains).
- ***description*** *(optional)*
  Optional description

Again, as can be seen in Figure 4.2, the provider is responsible for defining privacy options. Through this, they indirectly define which privacy policies are possible, i.e., they set the bounds in which users can choose their privacy policies. Most importantly, this determines which types of queries are possible on users' data. Different options could be made available to different users, e.g., to comply with privacy regulations in different jurisdictions or make more options available to paying users, though we do not discuss this further.

Example 4.5 provides a possible instantiation of a private sharing chain, where the aggregations sum, count, and mean are made available. To comply with this privacy option, only one such aggregation is made available each day (86400 seconds), and laplacian noise needs to be added to those aggregations. Queries need to add noise to the results of the above aggregations such that event-level DP with $\epsilon = 2$ and $\delta = 0$ is not violated.

Example 4.5 (Privacy Option)**.**
name*: "Differential Privacy"*
chain*: "Private Sharing"*
transformations*:* $[$
$($*"TimeRes"*$, \{86400, [$*"sum"*$,$*"count"*$,$*"variance"*$]\}),$
$($*"PertDP"*$, \{$*"event-level"*$, 2, 0,$*"laplace"*$\})$
$]$

Example 4.6 (Privacy Option)**.**
***name**: "Heart Rate Buckets"*
***chain**: "Private Sharing"*
***transformations**:* [(*"RangeRed"*,$\{60, 100\}$), (*"Bucket"*, $[\{0, 40\}, \{40, 60\},$ $\{100, 130\}, \{130, 160\}, \{160, 200\}]$)]
***appliesTo**:* [*heartRate*]

Such transformation chains with specific parameter selections may not be suitable for all value streams of an application. For example, we may define a chain that defines buckets for heart rates. These buckets will likely not fit any other value stream. For such situations, we have the "appliesTo" argument. By specifying "heartRate" for a correspondingly named value stream, the privacy option can only be applied to the value stream schema named "heartRate" (see Example 4.6).

More generally, since the application directly implies the data schema, there is not a lot that a provider can choose. For the privacy options this is different. Here, the provider influences which kinds of analyses are possible on top of data streams by setting corresponding privacy options.

So far, we have discussed how privacy options apply to regular value streams. For metadata streams, the same rules hold, with one exception: Only privacy options featuring deletion and metadata chains may be chosen. Since metadata streams are assumed to update infrequently, many chains that make sense for regular value streams would not work as expected for metadata streams. In addition, supporting arbitrary matching formulas is impossible under all transformations, e.g., if some metadata is bucketed, comparing this data to a fixed value that lies in the middle of the bucket could produce any result.

Privacy Policy Privacy options, as discussed above, constitute the main building blocks for privacy policies. For each value stream, a privacy policy defines a set of privacy options that can be applied to this value stream. Such a privacy policy can now be chosen by users directly or set by experts based on users' privacy preferences (see Figure 4.2). Even if a policy is automatically chosen based on users' preferences, the policy must be materialized (i.e., the privacy policy is in a form compatible with the API we describe in this section), which can then be used, e.g., for audits. Formally, we define a privacy policy as a list of *privacy policy parts*:

Definition 4.7 (Privacy Policy Part)**.**

- ***valueStreams*** *(required)*
  A nonempty list of names of value streams to which the options below apply.
- ***privacyOptions*** *(required)*
  A nonempty list of names of options, which can be applied to the value streams defined above. Each privacy option in this list needs to be compatible with all value streams, based on each options' "appliesTo" attribute and taking into account the restriction that only certain chains can be applied to metadata value streams.

Example 4.8 shows a possible scenario where a privacy policy part allows two privacy options (and with this transformation chains) named "Heart Rate Buckets" and "Differential Privacy" on a value stream with the name "heartRate".

Example 4.8 (Privacy Policy Part)**.**
valueStreams*: "heartRate"*
privacyOptions*: ["Heart Rate Buckets","Differential Privacy"]*

To be considered complete, each privacy policy needs to specify privacy options for each value stream:

Definition 4.9 (Privacy Policy)**.** *A privacy policy for a particular data schema consists of privacy policy parts, such that each value stream contained in this data schema (including metadata streams) is part of exactly one privacy policy part.*

A part that is implicitly defined by privacy policies is the privacy budget, which will become important in Section 4.3. If we have transformations involving differential privacy, it is natural to think of privacy budgets in terms of ϵ and δ, restricting how much personal data can be leaked. Running a query then consumes some of this budget, which is one of the most critical constraints regarding which data can be used for queries.

To have a unified model (which we will use for our privacy planner), we assign a budget to all chains, either explicitly (for differential privacy) or implicitly (for other chains). In principle, we can assign chains (and therefore privacy options) without differential privacy any budget. It is, however, essential to appropriately define the cost in terms of this budget for queries. If the query includes an aggregation, this query will consume the whole budget (to prevent differencing attacks). If it does not include aggregation, we only need to consume a tiny amount, such that no aggregation is possible anymore.

Using this assignment from budgets to privacy options as a basis, we can then assign each value stream a budget based upon which privacy options were assigned to this value stream by the privacy policy. However, we leave a detailed discussion of this to Section 4.3

Another open question is what precisely the effect of privacy groups is. We have defined this as follows: Two (or more) value streams in a privacy group share their privacy budget (the initial budget is the minimum of any value streams' budget). Therefore, any privacy budget consumption via one value stream uses up (part of) the budget for all value streams in the group.

Case Study In the following, we present a case study of a hypothetical fitness app. The goal is to highlight how to define *data schema*, *policy options*, and *privacy policy* for such an application and how the different parts of the user-centric privacy design interrelate.

Data Schema:

EventStream1

- **name**: "location"
  type: $(float, float)$
  range: $([-90, 90], [-180, 180])$

- **name**: "dataSource"
  type: $enum$
  metadata: $true$
  symbols: ["gps", "WLAN", "hybrid"]

EventStream2

- **name**: "heartRate"
  type: $long$
  privacyGroup: 1
  range: $[0, 250]$

- **name**: "stepsPerMin"
  type: $long$
  privacyGroup: 1
  range: $[0, 300]$

UserLevelMetadata

- **name**: ”weight”
  type: $long$
  metadata: $true$
  range: $[0, 250]$

- **name**: ”age”
  type: $long$
  metadata: $true$
  range: $[0, 130]$

In this data schema, we have a total of six value streams. Three of those value streams are metadata streams and can be used by the provider to filter users based on their values. Each stream forms its individual privacy group, except ”stepsPerMin” and ”heartRate”, which are in a joint privacy group since a correlation between heart rate and steps per minute is at least plausible. The location has a tuple data type $(float, float)$ which can store latitude and longitude, while ”dataSource” has $enum$ as data type, and either ”WLAN”, ”GPS”, or a hybrid approach as possible options. The other streams all have the $long$ data type and defined ranges to support differential privacy.

Privacy Options:

- **name**: ”Private”
  chain: ”Deletion”
  transformations: [”RedField”]
  appliesTo: [”metadata”]

- **name**: ”Differential Privacy”
  chain: ”Private Sharing”
  transformations: [(”TimeRes”,$(86400, [”sum”, ”count”, ”variance”])$), (”PertDP”, (”user-level”, $2, 0$,”laplace”))]

- **name**: ”Daily Aggregates”
  chain: ”Private Sharing”
  transformations: [(”TimeRes”, 86400)]

- **name**: ”Heart Rate Buckets”
  chain: ”Private Sharing”
  transformations: [(”RangeRed”,$\{60, 100\}$), (”Bucket”, [$\{0, 40\}, \{40, 60\}, \{100, 130\}, \{130, 160\}, \{160, 200\}$])]
  appliesTo: [heartRate]

- **name**: ”Shift”
  chain: ”Metadata”
  transformations: $[(Shift, 4)]$

- **name**: ”Public”
  chain: ”Public”
  transformations: $[UnmodRelease]$

The provider offers six privacy options, which are ordered roughly according to the privacy protection provided. Most of them can be applied to any value stream, except ”Private” (which can only be used in conjunction with one of the three metadata streams) and ”Heart Rate Buckets” (which can only be applied to the value stream named ”heartRate”)

Privacy Policy:

- **name**: ”location”
  policies: [”Differential Privacy”]
- **name**: ”dataSource”, ”weight”
  policies: [”Public”]
- **name**: ”heartRate”
  policies: [”Differential Privacy”, ”Heart Rate Buckets”]
- **name**: ”stepsPerMin”
  policies: [”Differential Privacy”, ”Daily Aggregates”]
- **name**: ”age”
  policies: [”Shift”]

Note that the above example is only one of many possibilities of how a privacy policy could be defined, given the above data schema and privacy options. In this policy, all non-metadata value streams can be accessed using the ”differential privacy” privacy option, while the ”heartRate” and ”stepsPerMin” streams can additionally be accessed using the privacy options ”Heart Rate Buckets” or ”Daily

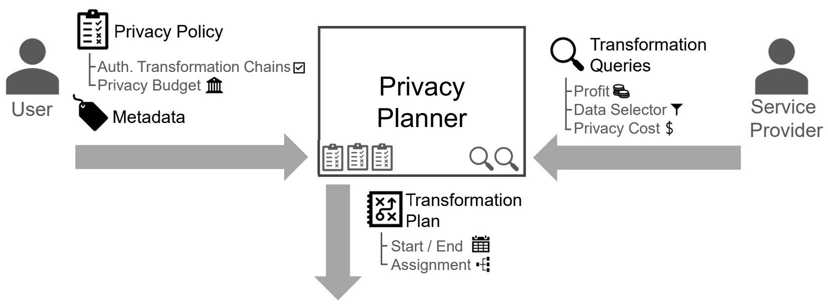


Figure 4.4: A closer look at inputs and outputs of the privacy planner.

Aggregates" respectively. Metadata is either public (dataSource), i.e., no restrictions apply, or shifted by a certain amount.

4.3 Privacy Planner

Following the discussion from Section 4.2, we now have a system that includes a set of event and value streams (according to an application-wide data schema) as well as a corresponding privacy policy for each stream. Each of these privacy policies defines a privacy budget for each value stream, as well as constraints regarding what sequences of transformations (i.e., authorized transformation chains) can be executed on a specific value stream. Finally, each user has some metadata (represented as slowly changing metadata value streams). Figure 4.4 provides an overview.

The provider can now specify a set of (transformation) queries, which signify what kind of analyses (i.e., transformation chains) they want to run on users' data. Conceptually, we think of a large set of small transformations to bring data into a privacy-compliant view. As an example, consider a set of users whose privacy policy requires the provider to make aggregations over at least 100 users. The provider may now want to aggregate over 1000 users. In this case, it makes sense to issue ten queries over 100 users each. On the resulting policy-compliant view, these ten queries can then be combined. However, also arbitrary other processing of these ten results is possible without violating any privacy policies.

This approach is essential for queries that do not include differential privacy, as strictly more information can be obtained without violating privacy policies. For queries with differential privacy, results that aggregate over ten times more people will typically also have correspondingly less noise. However, it is still not a disadvantage to run multiple small queries instead of a large one: The more considerable noise from the more minor queries will cancel out in expectation.

We assume that each transformation query has a specific profit (the value a specific query has for the provider), a cost in terms of privacy, and certain conditions in terms of metadata. Conditions on metadata (called ”Data Selector” in Figure 4.4) refer to filtering which streams can take part in a transformation chain (like the SQL ”WHERE” clause). Profit gives the provider control over which transformation chains they want our privacy planner to prioritize. For example, a provider may want to run 20 different transformation chains, and of each transformation chain a sizeable number. Additionally, they prefer running a small number of each transformation chain than a more significant number of only a single transformation chain if not enough streams are available. In essence, the provider can express such complicated preferences by appropriately defining profits. Finally, the privacy cost is the cost in privacy that executing a transformation chain has on users' value streams (or privacy groups).

Conceptually, the privacy planner takes privacy policies and queries and computes a transformation plan. This transformation plan contains an assignment of users' data streams to queries, start and end dates, which indicate how long the current matching is valid. It considers, among other constraints, the privacy cost of queries on the privacy budget of value streams and groups and specific requirements regarding user metadata as specified by the data selector. Finally, the privacy planner takes profits into account to determine which queries to prioritize. The goal is to maximize the sum of profits for queries in the transformation plan while respecting all users' privacy policies. Our privacy planner then allows near-optimal assignments (regarding the sum of profits) of users' data streams to providers' queries.

A component similar to the privacy planner is a requirement for any system that adopts a user-centric privacy design to extract utility safely from data while respecting users' privacy policies. For example, in Zeph [4], this component is called the policy manager. The policy manager uses a simple greedy approach to assign data streams to queries. We improve upon their design by introducing query profits and formulating the problem of computing a transformation plan as a constrained optimization problem. We consider an offline (or batch) approach, where we run the privacy planner after collecting a set of queries and data streams. Since this optimization can be expensive, we can include some slack (assign more users than needed) to a transformation to prevent it from failing as soon as a single user drops (which can happen with a high likelihood for aggregations including several hundred users). On the other hand, if new streams join, we can also add them heuristically without completely rerunning the optimization, possibly to transformations who recently ”lost” streams. To still not get too suboptimal results, we could set a threshold regarding how many users drop and join and rerun the complete optimization as soon as we pass that threshold.

The transformation plan produced by our privacy planner can then be used to compute a privacy-compliant view of the data (see Figure 4.1).

Towards the goal of presenting the inner workings of our privacy planner, we will first present how the provider precisely can specify queries. Given this specification, we present in detail what kinds of constraints and problems arise towards fulfilling these queries as well as possible. Finally, we present two approaches to solving this problem: A greedy, heuristic approach and an integer linear program (ILP) approach, generating (near) optimal transformation plans.

4.3.1 Expressing Queries

In this section, we define how queries can be expressed, and with this, define the possible set of queries considered in this work. The challenge is to provide both an expressive schema that allows many different queries while at the same time not creating a situation for the privacy planner that cannot be solved efficiently. Our query schema is a careful compromise between being expressive while supporting an efficient privacy planner. Recall that privacy policies describe which chains of privacy transformations respect the users' privacy preferences. Unsurprisingly, we opt for a query schema that follows a similar structure to simplify the job of the privacy planner. The query schema consists of required and optional attributes and defines a transformation chain to be executed.

Similar to how we described the supported privacy transformations in the privacy design in Section 4.2, we first define which kinds of transformations with which parameters are available in our system. The difference is that in a query, we specify a transformation chain executed on data (and not a ”minimum” as for privacy options). As a result, we require slightly different arguments. In addition to potentially changed parameters, we also define when a transformation that is part of a query is compatible with its counterpart from a privacy option. This compatibility between individual transformations will become important later when we define if a query and a privacy option are compatible.

- Range Redaction: *RangeRed*

  Here, we specify the same arguments as in the corresponding privacy option: The min and max of the redacted range. It matches its counterpart in a privacy option if and only if the redacted ranges are the same.

  - min: Defines the minimum of the redacted range (included).
  - max: Defines the maximum of the redacted range (not included).

- Tokenization: *Tokenize*

  Again takes the same arguments as its counterpart used in Section 4.2, and only matches it if the instantiated arguments are the same.

  - $func$: A function that is applied to every value that is to be tokenized.

- Shifting: *Shift*

  Again, it has the same arguments as its counterpart and matches it if the instantiated arguments are the same.

  - $size$: defines that the offset is chosen uniformly at random from the range $(-size/2, size/2)$, where $-size/2$ and $size/2$ are not included.

- Perturbation: *Pert*

  Again takes the same arguments but matches its counterpart if the added noise by the query is at least as big as specified in the privacy option.

  - $size$: defines that the offset is chosen uniformly at random from the range $(-size/2, size/2)$, where $-size/2$ and $size/2$ are not included.

- Differential Privacy Perturbation: *PertDP*

  For differential privacy perturbation, the arguments look very different: DP-$notion$ and $noiseType$ are the same as for privacy options, but the arguments ϵ and δ have been replaced by four new arguments: $epsPerRes$, $deltaPerRes$, $epsTotal$, and $deltaTotal$.

  The first two arguments control how much budget is used per generated result. These two arguments are enough information for event-level differential privacy, but for user-level differential privacy, we also need to specify how much privacy budget the query needs in total for privacy planner. For user-level differential privacy, dividing the total budget by the used budget per results (for ϵ and δ separately, and then take the minimum) then gives the total amount of how many results the query will return until the budget is used up. Note that this is a consequence of sequential composition (Theorem 2.7): If we generate k results and each of them has a differential privacy cost of (ϵ, δ), the total privacy cost is $(k \cdot \epsilon, k \cdot \delta)$.

  This transformation matches its counterpart only if multiple conditions are fulfilled: First, DP-$notion$ and $noiseType$ need to be the same. If we have event-level differential privacy, enough ϵ and δ budget needs to be available to accommodate $epsPerRes$ and $deltaPerRes$. For user-level differential privacy, the same holds true but with $epsTotal$, and $deltaTotal$ as costs.

  - DP-$notion$: event-level or user-level.
  - $epsPerRes$: How much epsilon should be used for each generated result.
  - $deltaPerRes$: How much delta should be used for each generated results.
  - $epsTotal$: How much epsilon should be used in total (for user-level dp).
  - $deltaTotal$: How much delta should be used in total (for user-level dp).
  - $noiseType$: What type of noise is used, e.g. gaussian or laplacian. Note that not all types of noise can be used to achieve all combinations of ϵ and δ.

- Bucketing: *Bucket*

  This transformation now again takes the same arguments as its counterpart for privacy options and only matches if the instantiated arguments are the same.

  - $buckets$: A list of individual bucket definitions. Each such definition contains two numbers min and max, which means that all values in the range (min, max) should be assigned to this bucket (including min, not including max). Optionally, a $label$ can be used to name a bucket. For multidimensional data types (e.g., location coordinates), multidimensional buckets can be set.

- Time Resolution: *TimeRes*

  While time resolution again takes the same arguments, it matches a privacy option only if $nSeconds$ is instantiated higher than or the same as in the option, and each aggregation function in $AggrFuncs$ is also part of the instantiated argument in the privacy option.

  - $nSeconds$: How many seconds the range of time that data should be aggregated over has.
  - $AggrFuncs$: Exactly which aggregate statistics are released (e.g., count, sum, variance).

- Population Resolution: *PopRes*

  Again, the arguments are the same. However, it matches only if each aggregation in $AggrFunc$ is contained in the $AggrFunc$ of the privacy option.

Note that $nPeople$ does not play a role in determining whether or not an option's transformation matches. If the query specifies fewer people than the privacy option, we can assign as many people as demanded by the privacy option and still run the query.

  – $nPeople$: How many people should be aggregated over at least.
  – $AggrFuncs$: Exactly which aggregate statistics are released (e.g. count, sum, variance...).

Note that field redaction is not a transformation available for queries, as just redacting data would not make much sense in a query.

In the above, we have always specified exactly when a transformation specified in a query matches a transformation specified in a privacy option. However, we have not yet explained when a transformation chain specified in a query matches a transformation chain specified in a privacy option. We define this as follows: For a transformation chain of a query (query transformation chain) to match a transformation chain in a privacy option (option transformation chain), each transformation in the option transformation chain must have a match (as defined above) in the query transformation chain, i.e., the same transformation with appropriate parameters.

With this, we can now define the schema for a query as follows:

- **name** (required)

  Which name the query should have.

- **inputSchema** (required)

  Define which value stream schema should be used as input. The chosen name must correspond to the name of a value stream in the data schema of the same application. Note that this restricts queries to operate on only one kind of value stream. However, since arbitrary post-processing is allowed, data from different value streams may still be combined.

- **chain** (required)

  Which chain (see Section 4.2.2) is used.

- **transformations** (required)

  Which transformation chain is used to transform data. It needs to conform to the type of chain defined above, with parameters as defined earlier in this section (and often different from the parameters used in privacy options).

- **conditions** (optional)

  Define a condition on any metadata streams similar to a SQL ”WHERE” clause. Then, it will only take into account streams that fulfill these conditions.

- **numberAndProfit** (optional)

  A list that defines two things at once: How often should this query execute the defined transformation chain and the profit for each execution. E.g. $[(10, 300), (5, 100)]$ means we want to execute the chain in total $10+5 = 15$ transformations. We assign the first 10 transformation chains a profit of 300 each and the other 5 a profit of 100 each.

  Note that we will try to optimize the sum of profits of all executed transformation chains in the following. If the goal is to run as many transformation chains from as many different queries as possible, the idea is to assign a high profit for a few chains and lower profits for others. Since the profit is not weighted by population constraints (from PopRes), the assignment should assign higher values to queries with high population requirements. Consider the following example: We have two queries, one involving an aggregation over 100 users and one over 1000 users. In all other regards, they are the same, which also means their privacy cost is the same for each assigned stream. Accordingly, the second query (over 1000 users) uses ten times more privacy budget. Suppose we assign both queries the same profit. In that case, the query over 1000 users has a much lower profit to privacy cost ratio, and our privacy planner will likely prioritize the query with 100 users.

  If this attribute is not set, the default is $[(1, 1)]$, i.e., we want to execute the chain one time and assign it a profit of 0.1.

Building on the case study presented in Section 4.2, a possible query defined using the above schema could look as follows:

Example 4.10 (Query Schema)**.**

- ***name***: *"dailyHeartRate"*
  inputSchema: *"heartRate"*
  chain: *"Private Sharing"*
  transformations:
  $[($*"RangeRed"*$, (60, 100)),$
  $($*"Bucket"*$, [\{0, 40\}, \{40, 60\}, \{100, 130\}, \{130, 160\}, \{160, 200\}]),$
  $($*"TimeRes"*$, (86400, [sum, count, variance])),$
  $($*"PopRes"*$, (100, [sum, count, variance])),$
  $($*"PertDP"*$, (user\text{-}level, 0.01, 0, 1, 0, laplacian))]$
  conditions: *residentialRegion==*"Zurich" $\&\&$ *age*$>= 50$
  numberAndProfit: $[(10, 400), (200, 50)]$

- ***name***: *"weeklySteps"*
  inputSchema: *"stepsPerMin"*
  chain: *"Private Sharing"*
  transformations: $[($*"TimeRes"*$, [604800])]$
  conditions: *age*$<= 65$
  numberAndProfit: $[(100, 3), (1000, 1)]$

4.3.2 Problem Statement

This section outlines the challenges of matching queries and users' streams in a user-centric design. The goal is to maximize the sum of profits (from executable queries) while not violating any constraints set by privacy policies or by queries regarding users' metadata. After discussing the challenges, we discuss two different approaches to solve the problem: A heuristic approach and an integer linear program (ILP) approach.

C1: Privacy Policy Compatibility The first challenge is to determine, based on each users' privacy policy, which queries can use a users' data. In other words, we need to make sure that the result generated by our privacy planner complies with all privacy policies. However, we do not include population constraints here, i.e., if a privacy option only allows aggregations over specific amounts of people. We treat this challenge separately since the approaches to this challenge are pretty different from respecting population constraints. We also do not take privacy budgets into account in this challenge and instead set this as the next challenge.

C2: Managing Privacy Budgets The second challenge is managing privacy budgets. For differential privacy, this implies deciding which queries get allocated what amounts of ϵ and δ. However, for queries and privacy options that do not involve privacy options, we often need to limit how often data can be reused before reconstruction attacks become feasible. Unifying these different kinds of budgets is part of this challenge. Else we would end up with data silos that have to be queried separately.

C3: Taking Metadata into Account The third challenge is now to conform to a query's ”conditions” attribute, i.e., the data selector from Figure 4.4.

C4: Handling new users and dropouts Users could leave and join the system at any time. The fourth challenge is to ensure that our privacy planner takes this into account, i.e., we do not want to rerun the complete optimization each time a new user leaves or joins. This challenge also includes changing metadata, which can make a user suddenly infeasible for a query.

C5: Respecting Population Constraints The fifth challenge is given by actually respecting the population requirements of both any participating streams and queries. For example, suppose a query only needs 100 users, but an otherwise compatible (see C1) privacy option demands 1000 users. In principle, this query can still use data under this privacy option but needs to assign at least 1000 users if a user with these requirements is included. Whether or not it is worth it to therefore include such users depends on the specific situation. For example, suppose there are almost exclusively users in the system that allow queries only if aggregations include at least 1000 people. In that case, it might not be possible otherwise to run the query.

C6: Maximizing Profits Finally, the last challenge is to respect the above challenges while maximizing the sum of profits of executed queries. For each transformation chain, we use an all-or-nothing principle: Either the transformation chain runs and therefore generates a profit as specified in numberAndProfit, or it does not run and generates 0 profit.

Addressing the challenges Seeing all those challenges, we will now address how we approach them in our privacy planner.

First, note that challenges C1 and C3 can be addressed by ”filtering”: For each stream and each query, we can independently check whether they match metadata and privacy policies. We cannot check C2 (budget constraints) and C5 (population constraints) this way since whether or not we fulfill these conditions also depends on the assignment of other streams. If a stream and query pass these checks, we will describe them as *compatible* in the following.

As explained above, a query can match a users' value stream if that value stream is assigned a matching privacy option. This condition means that each transformation that is part of this privacy option is also part of the queries transformation chain, and these corresponding transformations have compatibly instantiated arguments. As an example, consider a query aggregating data over one hour. We can use data that needs to be aggregated over 100 seconds, but not data that requires daily aggregations. On the other hand, metadata can be checked by simply evaluating the conditions set in the query on each user's current metadata.

This discussion immediately brings us to challenge C4: What happens if users drop out, or their metadata no longer matches queries original specifications. We propose three mechanisms to deal with this challenge: First, we can always assign more users to a query than it requires (e.g., 110 users if the query requires an aggregation over 100 users). This way, we have some slack if users drop out or become ineligible. Second, based on the heuristic approach discussed later, we can prioritize assigning new users or users with changed metadata that have become ineligible to queries where many users dropped out. Finally, if the metadata changes predictably (e.g., the age will only change once a year), we can refrain from assigning streams that will become ineligible for a specific query soon.

For the second challenge (C2), we first present how to unify the privacy budget for differential privacy options and other options. As already explained, for differential privacy, the budget is given by particular values for ϵ and δ. For other privacy options, we set this budget to an arbitrary, fixed number (e.g., one million, in both dimensions). This way, all privacy options imply a two-dimensional budget, allowing us to unify budget management. Using the budget implied by each privacy option, the privacy policy then assigns each value stream a budget. Suppose any privacy option that applies to a particular value stream has differential privacy included in its chain. In that case, the budget will be the maximum differential privacy budget in terms of ϵ and δ. Otherwise, it will just be this fixed number again (in both dimensions).

How much budget is consumed now depends on the query. Table 2 summarizes the results, which we will discuss below. Depending on whether or not queries or privacy options include differential privacy, different cases are possible.

- A differential privacy query applied to value streams with a differential privacy option will consume exactly the stated budget in terms of ϵ and δ. For user-level differential privacy, this deduction corresponds to $epsTotal$ and

Costs	Value Stream w/ DP	Value Stream w/o DP
Query w/ DP and w/ aggr.	as specified in Query (ϵ and δ)	entire budget
Query w/ DP and w/o aggr.	as specified in Query (ϵ and δ)	$(0.1, 0.1)$
Query w/o DP and w/ aggr.	entire budget	entire budget
Query w/o DP and w/o aggr.	$(0.1, 0.1)$	$(0.1, 0.1)$

Table 2: The cost of applying a query to a value stream

$deltaTotal$, while for event-level it is given by $epsPerRes$ and $deltaPerRes$, as described above.

- A query without differential privacy applied to a value stream without differential privacy option will consume all available budget if the query includes an aggregation (over people or time, to prevent differencing attacks). All available budget means all budget that is available initially, i.e., no other query may use this data. On the other hand, if no aggregation occurs, only a tiny but nonzero budget is subtracted (e.g., 0.1 in both dimensions). In this case, there is no need to prohibit other queries on the same data since differencing attacks are not of concern if no aggregation is taking place.
- A query with differential privacy applied to a value stream without differential privacy (i.e., no option features differential privacy) will behave the same as if the query did not have differential privacy in terms of budget. It will, therefore, consume the entire budget to guard against differencing attacks if aggregation is taking place. Otherwise, it just consumes a tiny number (e.g., 0.1).
- Finally, a query without differential privacy could match a value stream with differential privacy. This situation can only happen if at least two privacy options are assigned to the value stream, one with differential privacy (which cannot match a query without differential privacy) and one without, which matches the query. In this case, all available budget is consumed if aggregation is taking place, and otherwise a tiny number (e.g., 0.1).

This discussion leaves challenges C5 and C6: How to respect population constraints and maximize profits. Since our solutions to these questions differ in our two approaches, we will discuss these challenges in the respective sections.

Abstract Problem Statement Towards finding optimal solutions, following from the above discussion, we will now present a formal problem statement that considers all the challenges as defined above. The problem of maximizing the sum of profit becomes a constrained optimization problem. In the following, we denote with "equal profit group" (epg) a set of transformation chains that are part of the same query and have the same profit. E.g., a query where the numberAndProfit attribute is set to $[(10, 100), (30, 20)]$ has two equal profit groups: One consisting of 10 transformation chains, each giving (potentially) a profit of 100, and another one consisting of 30 transformation chains, where each one can generate a profit of 20.

Definition 4.11 (Formal Problem Statement). *Let $S = \{s_1, ..., s_n\}$ be a set of streams and $EPG = \{e_1, ..., e_k\}$ the set of all all equal profit groups. Each $e_j \in EPG$ has a associated profit $p(e_j)$ and $par(e_j)$ many transformation chains are part of e_j. A compatibility matrix $C \in \{0, 1\}^{n \times k}$ defines if a stream s_i is compatible with a equal profit group e_j: s_i and e_j are compatible if and only if $(C)_{i,j} = 1$. Additionally, each equal profit group e_j has a minimum population requirement $pop(e_j)$ and each stream s_i has a minimum population requirement $pop(s_i, e_j)$ that depends on the query e_j. Finally, each stream s_i has a budget $budget(s_i)$ and there is a matrix $B \in \mathbb{R}^{n \times k}$ where the entry $(B)_{i,j}$ defines how much budget e_j would use from $budget(s_i)$, if s_i is assigned to e_j*
The goal is now to find an assignment of streams s_i to equal profit groups e_j, where a stream can be assigned to multiple queries and multiple streams to one query (i.e. a binary relation $A \subseteq S \times EPG$), such that $\sum_{j=1}^{k} y_j * p(e_j)$ is maximized while complying with the following constraints:

1. *Number of transformation chains per epg: $\forall j \in [k] : 0 \leq y_j \leq par(e_j), y_j \in \mathbb{Z}$, i.e. only $par(e_j)$ many parallel executions count towards the optimization goal*
2. *Stream and Query Population Requirements: For each epg e_j, we can subdivide the streams assigned to e_j into y_j groups $G = \{g_1, .., g_{y_j}\}$, such that each group $g \in G$ has a size $|g| \geq pop(e_j)$ and for each $s_i \in g$ we have $|g| \geq pop(s_i, e_j)$. This subdivision also needs to be part of the output (i.e. it is not enough that this subdivision exists).*
3. *Compatibility requirement: $\forall (s_i, e_j) \in A : (C)_{i,j} = 1$, i.e. all assignments are compatible*
4. *Budget Requirement: $\forall s_i \in S : \sum_{j=1}^{k} \left(\mathbb{1}_{(s_i,e_j) \in A} \cdot (B)_{i,j} \right) \leq budget(s_i)$, i.e. we use at most the given budget for each stream s_i*
5. *Uniqueness requirement: Let $EPG(q)$ be all equal profit groups belonging to some query $q \in Q$, where Q is the set of queries. $\forall s_i \in S, q \in Q$: Can only assign s_i to at most one $e_j \in EGP(q)$*

It is important to note that we only consider a one-dimensional budget for simplicity, which means we are limited to $\delta = 0$ for differential privacy. Such an extension to two dimensions would be trivial: We introduce a second budget akin to the first one, with the same constraints (but different values for many instantiations). However, two-dimensional budgets may not even be needed even for $\delta > 0$: According to [20], if we want to enforce global differential privacy guarantees, tracking ϵ is enough since computations are much more sensitive to ϵ than to δ. The reason is that the noise added to enforce differential privacy scales linearly in $1/\epsilon$ and at most logarithmically in $1/\delta$ for common types of noise. Therefore, if we enforce certain levels of ϵ, enforcing particular levels of δ is possible with only small amounts of added noise, which barely impacts the result.

Additionally, note that a stream in this problem corresponds to a value stream of a particular user only if there are no privacy groups specified in the corresponding data schema (remember that each value stream forms its own privacy group if nothing different is specified). If there are privacy groups larger than one, a stream in the above problem instead refers to all value streams of a specific user in a particular privacy group. These then share a budget (and in the above formulation, each stream has a separate budget). Finally, note that since queries only apply to single types of value streams, we can look at the above problem separately for each privacy group, which can be helpful for implementation purposes. In other words, if we have optimal or near-optimal results for each privacy group separately, we get optimal or near-optimal results for the global optimization problem.

In addition to the constraints expressed in Definition 4.11, we can assume that there will only be a few distinct Stream Population requirements, i.e., the set $\{pop(s_i, e_j) | s_i \in S, e_j \in EPG\}$ is small. While this does not impact the correctness of any solution to the problem, this assumption may help find faster solutions.

From this description, it should be clear that the problem is nontrivial to solve optimally in any nontrivial situation. Of course, if the queries partition the available users without much overlap, trivially assigning users to the only available query will deliver optimal results. Nevertheless, for a more realistic situation involving a more extensive set of queries and users being compatible with multiple queries, an optimal solution requires keeping track of all the above constraints while optimizing profits.

Virtual Queries Before discussing the two approaches, we need to introduce one additional concept: *virtual queries*

As stated above, we can assume that there will be only a few distinct stream population requirements in the system. Let PR be the set of stream population requirements in the system (e.g. $\{100,500,1000\}$). Let e_j be some equal profit group belonging to a query with query population requirement $x = pop(e_j)$. Then the set of virtual queries belonging to e_j is given by $\{e_j, e(max(x,100))_j$, $e(max(x,500))_j$, $e(max(x,1000))_j\}$, where $e(y)_j$ mean the transformation chain of e_j was modified to include a query population requirement y (instead of $x = pop(e_j)$ as before). So, in essence, we have several virtual queries corresponding to the different stream population requirements for each equal profit group.

4.3.3 Heuristic Approach

Before we dive into an optimal solution formulated as an integer linear program, we present a greedy, heuristic approach. Solving integer linear programs is generally expensive. The goal of the heuristic approach is to deliver acceptable results with more performance (i.e., faster to execute and less memory used) for situations where running the optimal solution is not feasible. In addition, it will serve as a baseline in Section 6 to evaluate the performance of the near-optimal (in terms of the result) approach.

Abstractly, the heuristic approach proceeds by ordering virtual queries depending on how much profit they bring per assigned stream. As an example, a query that aggregates over 100 people and a profit of 100 would be equal to a query without aggregation over people and a profit of 1. In this example, both queries have a profit of one per assigned stream. In other words, we use the potential profit per assigned stream as a heuristic to decide on the order of queries. We then proceed to assign streams greedily to queries according to this ordering.

We now present the heuristic approach in more detail:

1. From the set Q of queries generate the set EPG of equal policy groups, and from this VQ, the set of virtual queries.

2. For each virtual query $vq \in VQ$, determine all compatible streams, additionally requiring that any stream population requirements are not larger than the query population requirement of vq.

3. For each virtual query $vq \in VQ$, calculate the profit per user by dividing the profit per transformation chain by the number of users.

4. Sort the virtual queries according to profit per user

5. Go through the sorted list of virtual queries, starting with the virtual query with the highest profit per user. For each virtual query vq, we do the following:

   (a) Determine how many compatible streams also fulfill the budget requirement and are not already assigned to another virtual query of the same query.

   (b) Determine how many streams m we can assign to this virtual query at most to make a profit still (taking into account streams already assigned to other virtual queries of the same equal profit group).

   (c) Add as many streams as possible to vq, without "wasting" any. More formally: If we have x available streams and vq has population requirement y, we add $min(m, x - (x \mod y))$ many streams. Each y consecutive streams together for an "execution group" (in the sense of groups $g \in G$ from Definition 4.11)

   (d) Adjust the budget of any streams used this way.

6. By mapping the streams and execution groups assigned to each virtual query vq back to the original query, we then have the desired mapping of streams to queries.

4.3.4 Integer Linear Program (ILP) Approach

Finally, we present our integer linear program (ILP) approach to solve Definition 4.11. While the heuristic algorithm should deliver acceptable results, this approach delivers optimal results, given the provider's preferences. In other words, it can make optimal use of privacy resources to generate as much value as possible for the provider.

Before stating the program, we will quickly state some assumptions required to understand all indices used in the ILP. We have a set of streams $S = \{s_1, ..., s_n\}$, a set of queries Q, a set of equal profit groups $EPG = \{e_1, ..., e_k\}$ and, for each $e_j \in EPG$, a set of virtual queries $SC(e_j) = \{vq_{j_1}, ..., vq_{j_{|SC(e_j)|}}\}$. We abuse notation by shortening $\forall i \in [n]$ to $\forall i$, $\forall j \in [j]$ to $\forall j$, $\forall q \in Q$ to $\forall q$, and $\forall j \in [k], \forall a \in [|SC(e_j)|]$ to $\forall j_a$ to keep the ILP as short as possible.

Definition 4.12 (Integer Linear Program)**.**

- ***variables:***
  - $\forall i, j_a : x_{i,j_a} \in \{0, 1\}$
  - $\forall j_a : y_{j_a} \in \mathbb{N}_0$
- ***objective:*** $max \sum_{j_a} y_{j_a} p_j$
- ***subject to:***
  1. $\forall i : \sum_{j_a} x_{i,j_a} \cdot c_j \leq B_i$
  2. $\forall j_a : \sum_i x_{i,j_a} \geq y_{j_a} \cdot pop(vq_{j_a})$
  3. $\forall i, j_a$ *where* $(C)_{i,j} = 0$ *or* $pop(vq_{j_a}) < pop(s_i, e_j) : x_{i,j_a} = 0$
  4. $\forall j : \sum_{a=1}^{|SC(e_j)|} y_{j_a} \leq par(e_j)$
  5. $\forall i, q : \sum_{\{j_a | e_j \in EPG(q)\ \&\& a \in SC(e_j)\}} x_{i,j_a} \leq 1$

We will now offer some explanations as to why this ILP solves our problem. Towards this goal, we will first clarify the meaning of the two types of variables. x_{i,j_a} is a binary variable, and signifies whether or not stream s_i was assigned to virtual query vq_{j_a}. y_{j_a} is a non-negative integer, which determines how many transformation chains of some virtual query run.

With this in mind, we can now explain the different constraints and how they relate to Definition 4.11.

- The first constraint (item 1) ensures that for each stream s_i, the set of assigned virtual queries s_i does not exceed the budget from s_i, i.e., it enforces the budget constraint (Definition 4.11, item 4).
- The second constraint (item 2) ensures that there are enough streams assigned to each virtual query vq_{j_a} to actually run y_{j_a} many transformation

chains. By definition of virtual queries, this solves the stream and query population requirements (Definition 4.11, item 2), together with the third constraint. We can get appropriate execution groups by arbitrarily partitioning streams assigned to a virtual query vq_{j_a} into y_{j_a} equally-sized partitions.

- The third constraint (item 3) both ensures that streams are only assigned to a virtual query if compatible with the query (Definition 4.11, item 3) and, together with the second constraint, ensures that stream and query population requirements are not violated (Definition 4.11, item 2). Of course, in an actual implementation, we would not add variables which can only take a single value (i.e., 0).
- The fourth constraint (item 4) enforces that we can only profit from at most $par(e_j)$ many transformation chains per equal profit group, i.e., any solution to this ILP under this constraint will comply with Definition 4.11, item 1
- The fifth constraint (item 5) finally makes sure that for each query q, each stream can only be assigned to at most one virtual query belonging to q, i.e. it enforces the uniqueness requirement from Definition 4.11, item 5

From this discussion follows that, while the ILP and the abstract formulation in Definition 4.11 differ in many regards, any solution to the integer integer program from Definition 4.12 implies that a solution to the problem from Definition 4.11 with equal target value exists and vice versa. Additionally, we can efficiently convert solutions between the two forms.

5 Implementation

This section gives an overview of the prototype. First, we lay out how it was implemented and present critical features. Then, we detail the interface and explain the program behavior on a higher level. Finally, we mention which additional limitations and assumptions are part of the prototype, different from our design.

We implement the privacy planner in Python 3.8.10 and support both the heuristic approach and the ILP approach. To solve ILPs, we use Gurobi 9.1.2 with an academic license and gurobipy as an interface between Python and Gurobi. We support the full set of queries and privacy policies as defined in Section 4. Users (with associated metadata and privacy policies) can be randomly generated in a separate module by specifying a set of metadata attributes (including the range of values) and privacy policies. For metadata, we used a custom distribution (see Table 4), while a privacy policy is assigned, independently from metadata, uniformly at random.

The data schema, queries, privacy options, and privacy policies must be defined directly in the program itself. Constraints regarding metadata (e.g., only consider people over a certain age) are expressed using python lambda functions [29]. If the metadata is of the enum type, a corresponding class needs to be defined in python (see [28]).

Several options can be set using command-line arguments when starting the program. Most importantly, options exist to set the number of randomly generated users and use the heuristic approach, the ILP approach, or both. In addition, we can set how distant the ILP solution may at most be from an optimal solution (which is enforced by Gurobi) and how long the ILP optimization part is allowed to run at most. Another argument controls whether or not a random seed is chosen (i.e., whether or not the user generation is pseudo-random). Finally, we can set if we use the solution from the heuristic approach as a starting point in Gurobi or not. Some other options control output behavior, modify the set of Queries, and other minor modifications to the Gurobi solver exist.

Listing 1 provides an overview of how the program proceeds. In the following, we will add some additional information:

1. We load the set of queries and privacy policies that are specified in the program.
2. We randomly generate a set of users with associated metadata and privacy policies.

3. For all users and queries, we determine whether or not they are compatible (challenges C1 and C3 from Section 4.3).
4. If the heuristic approach is enabled, we run it as described and save the result.
5. If the ILP approach is enabled as well, we now build the ILP model. For this, we first initialize the model, add variables to it, set an objective, and add constraints. If the heuristic approach was run, we also add the heuristic solution to the model, which can be used as starting point for the optimization. Finally, we optimize the model and save the result.
6. As the last step, we compare the two approaches (if both were run) and generate any needed output.

```
queries, privacy_policies = init()
users = generate_random_users(privacy_policies)
virtual_queries = generate_virtual_queries(users, queries)

# calculate the compatibility matrix
compatibility = check_compatibility(users, queries)

if use_heuristic:
    heuristic_assignment = run_heuristic(virtual_queries,
                                         compatibility)

if use_ilp:
    ilp_model = init_ilp()
    ilp_model.add_variables(virtual_queries, compatibility)
    ilp_model.add_constraints(virtual_queries, compatibility)

    if use_heuristic:
        ilp_model.set_inital_solution(heuristic_assignment)

    # solve the ilp with gurobi
    ilp_assignment = ilp_model.optimize()

if use_heuristic and use_ilp:
    comparison = compare_assignment(heuristic_assignment,
                                    ilp_assignment)
```

Listing 1: Pseudocode which gives an overview of how the implementation proceeds

While we, therefore, define privacy policies without any user input, no significant changes are needed to support the case where users choose their privacy policy. We can extract all chosen policies in a first pass and then assign them to the users who chose them. This assignment can be implemented by overriding the module which currently generates random users.

In the artifact, we have made several assumptions to focus on the most significant issues. First, we assume that we only have a single type of value stream. However, this assumption still covers the case with multiple types of value streams since we can run the program for each type separately, as long as these types of value streams are independent (as discussed in Section 4.3). In our design, this means we assume that there are no privacy groups. Although, an extension to include privacy groups would be pretty natural, as, in these, value streams of the same user just share a budget.

Additionally, we assume that there are no transformations done on metadata and that metadata does not change, even though this is supported by our design. Furthermore, we also assume a simplified cost model, where we only consider a one-dimensional budget (as in Definition 4.11). Furthermore, some differences exist regarding the cost of queries to Table 2. Queries with differential privacy always consume as budget ϵ, or else (no DP in the query) the entire budget of a stream if the query includes an aggregation over people, or otherwise (no DP, no aggregation over people) just 1 budget unit. Finally, no external API is available for in- or output, save for two exceptions: User generation (or input, if the data is not randomly generated) is implemented by calling a function from an external module, which can be replaced as needed. Additionally, results from the ILP solution and several statistics about program performance can be written to a file.

6 Evaluation

In the evaluation, we aim to answer the following three questions:

Q1: What advantages in terms of profits do we gain by using the ILP approach compared to the heuristic baseline, i.e., how much better are the assignments?

Q2: How scalable are the two approaches in terms of both runtime and memory?

Q3: What are the important factors that influence the runtime of the privacy planner?

6.1 Experimental Setup

We obtained all benchmarks by running our implementation using AWS Elastic Compute Cloud (EC2) on a z1d.2xlarge instance running Ubuntu Server 20.04 LTS. We run our privacy planner as a service and used Ansible to orchestrate the overall experiments.

Towards creating the benchmarks, as described in Section 4.1 and Section 4, we need a data schema, a set of privacy options and privacy policies, a set of users with according metadata and privacy policies, and a set of Queries. Since systems with user-centric privacy are not yet deployed in any significant setting, we carefully construct an application that includes very different types of queries, such that we have multiple interesting tradeoffs.

Finally, we passed a few parameters to Gurobi to modify its standard behavior. First, Gurobi allows setting a parameter that controls how far away an output solution is from optimality at most. We set this parameter to 0.05, meaning that any solution Gurobi found was at most 5% worse than the optimal solution for the input ILP. Setting this parameter even lower could mean finding an even better solution at the cost of increased runtime. In addition, we set Gurobi to focus on finding a feasible solution quickly, instead of trying to prove the optimality of already found solutions, which seemed to work well for our problem instance (see below). Finally, we arbitrarily set a time limit of 2.8 hours for the optimization, though this limit was never reached for our experiments (which involved up to one million users).

6.2 Problem Instance

The data schema consists of just a single value stream (heart rate) and two metadata streams (age and area). Each user belongs to one of five age groups and one of seven areas. Each user was randomly assigned an age group and an area based on the real data from Switzerland. Table 4 contains the specific numbers used towards creating this probability distribution. More information is included in the appendix Section A.1.

We include seven privacy options that include transformations such as differential privacy, bucketing, aggregation over time and/or aggregation over people (see Section A.1). From these, we define seven privacy policies (see Section A.1), which are roughly ordered according to which level of privacy they provide (from no access for the provider over only DP to full access for the provider). As per the assumptions made in Section 5, metadata is accessible without any restrictions for all privacy policies. For this evaluation, each user is assigned a privacy policy uniformly at random, independent of the metadata assignment.

Finally, in Table 3, we give an overview of the queries used in our benchmark. More information can be found in Section A.1 in the appendix. For the queries without DP, we assigned very high profits to a small number of transformation chains, medium profits to a bigger number of transformation chains, and low profits to many more transformation chains. For queries with DP, we just assigned low profits to a large number of transformation chains. The idea is that we want a few transformation chains of every query without DP to do certain analyses that depend on these queries. Suppose we have a lot of users available. In that case, it is no problem to run at least a small number of transformation chains of every query, so we do not have a particular preference anymore. Certainly, we do not want to leave data unused.

Query	Query Description	Has DP
Q1	Bucketed heart rate of users over 65 that live in the Zürich area and are at least 65 years old	☐
Q2	Bucketed heart rate of users that live in the Ticino area	☐
Q3.1	Hourly aggregations (sum, count and variance) of heart rate over 100 users aged at least 65 years	☐
Q3.2	Hourly aggregations (sum, count and variance) of heart rate over 100 users aged between 40 and 64	☐
Q3.3	Hourly aggregations (sum, count and variance) of heart rate over 100 users aged between 20 and 39	☐
Q3.4	Hourly aggregations (sum, count and variance) of heart rate over 100 users aged at most 19 years	☐
Q4.1	Daily aggregations (sum, count and variance) of heart rates over 100 users	☑
Q4.2	Weekly aggregations (sum, count and variance) of heart rates over 100 users	☑

Table 3: An overview of queries used for our microbenchmark

We chose this specific problem instance setup as the result of two main criteria. First, we wanted to include a range of common privacy transformations, both in the query and in privacy options and policies, to approximate what a problem instance could look like in an actual deployment. Second, we wanted to create a scenario featuring multiple difficult tradeoffs, as would be likely in real-world deployments where multiple actors want to run all kinds of analyses on data, but the privacy budget is too limited to fulfill all requests. In such a scenario, privacy management is the deciding factor for performance (in terms of profit), which is exactly what we want to evaluate. If our approaches perform well in such a demanding scenario, it is plausible that they will also perform well in less challenging circumstances.

6.3 Methodology

In general, for different parameters, we measured the runtime of (different parts of) our implementation, memory consumption, and profits of both the heuristic approach and the ILP approach.

We repeat the run for each parameter configuration five times; report mean values and the standard error of the mean. The only exception is memory consumption, where we instead plot the maximum amount of memory observed in any of the five repetitions, i.e., the observed worst case.

Any timing information was obtained using the ”time” module of python [30]. Profit measurements and the number of variables and constraints are directly available as a result of our implementation. We measure memory at a fixed interval during the program's runtime, using the ”systemctl” command available in Ubuntu Server 20.04 LTS.

6.4 Benchmarks

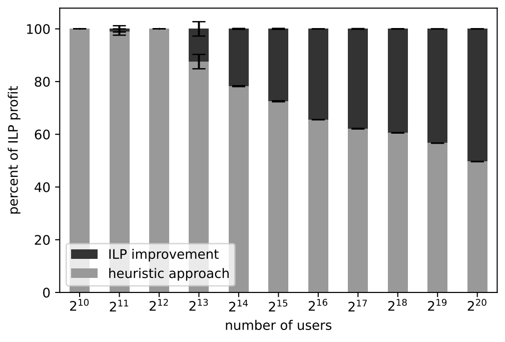
Question 1 (*What advantages in terms of profits do we gain by using the ILP approach compared to the heuristic baseline, i.e., how much better are the assignments?*)

In Figure 6.1a, we show which percentage of profit the heuristic algorithm achieved compared to the ILP approach, as we vary the number of users. In Figure 6.1b, we show the total profit when we vary the number of users (using a log-log scale). For 2^{12} or fewer users, the heuristic and the ILP approach result in almost the same profit. For more users, the profit ratio continuously gets larger, until for 2^{20} users, the ILP result improves the profit over the heuristic baseline by a factor of almost 2.

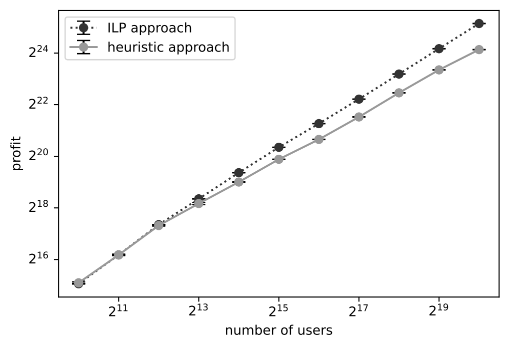
The experimental results show that for a small number of users, the ILP cannot significantly improve the profit over the heuristic approach. However, with increasing numbers of users, there is more room for the ILP to improve the result over the heuristic baseline.

Question 2 (*How scalable are the two approaches in terms of both runtime and memory?*)

We start with addressing memory usage. In Figure 6.2, we plot the maximum memory consumption when we vary the number of users. Figure 6.2a uses a linear scale, while Figure 6.2b has a log-log scale. We can see that the ILP approach always consumes more memory no matter the number of users. For $100K$ users the ratio is 6.8, for $500K$ users 8.4, and for one million users 8.2.

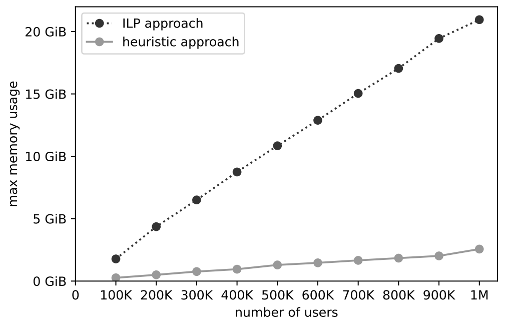


(a) Which percentage of the ILP profits the heuristic approach netted.

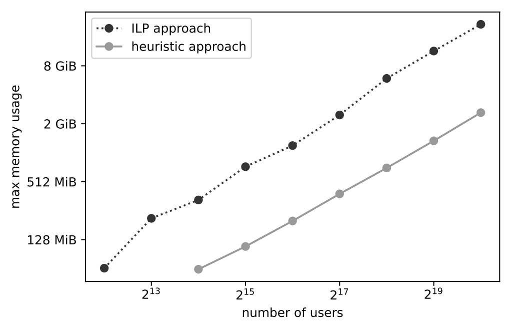


(b) total profits on a loglog scale

Figure 6.1: The profits of heuristic and ILP approaches compared for different numbers of users.



(a) linear scale



(b) loglog scale

Figure 6.2: The maximum memory requirements of our ILP and heuristic approaches compared.

However, both approaches' memory consumption scales roughly linearly in the number of users. This behavior does not come as a surprise: since all python data structures, as well as the size of the ILP model (number of variables and constraints, see Figure 6.3) scale linearly with the number of users, the memory consumption of the whole program should as well. Due to this linear scaling, we can therefore consider both approaches scalable to larger deployments, yet the ILP still uses considerably more memory.

After looking at the memory consumption, we now consider runtime. Initially, it is much less clear how the runtime scales with the size of the problem for our ILP approach. It is well-known that integer linear programs can solve NP-hard problems and run exponentially in the worst case. However, for many practical

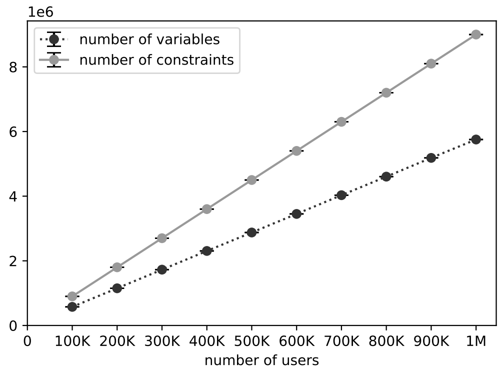


Figure 6.3: The number of variables and constraints of the ILP depending on the number of users.

problem instances, an (approximately) optimal solution can be found quickly with Gurobi or other commercial ILP solvers.

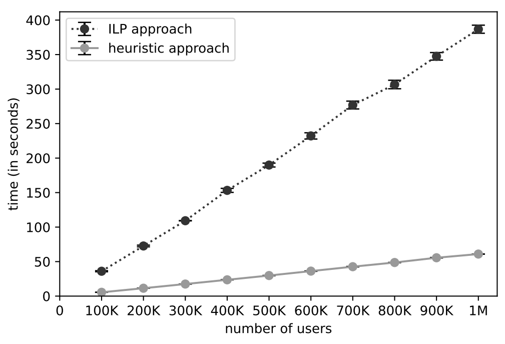
In Figure 6.4, we show the runtime when we vary the number of users, again using a linear scale (Figure 6.4a) and a log-log scale (Figure 6.2b). We can see that no matter the number of users, the ILP approach always takes longer. For $100K$ users the ratio is 6.5, for $500K$ users 6.4, and for one million users 6.3.

Nevertheless, as for memory, both approaches scale linearly. Note that the transformation manager is designed to run offline workloads, i.e., should not be placed on the critical path. With this in mind, both approaches are certainly feasible also in terms of runtime. That the heuristic algorithm scales linearly with the number of users is clear from the description in Section 4.3. However, as described above, it is initially unclear why the ILP approach also scales linearly with the number of users. To address this, we next want to look into what parts of the approaches contribute the most to runtime.

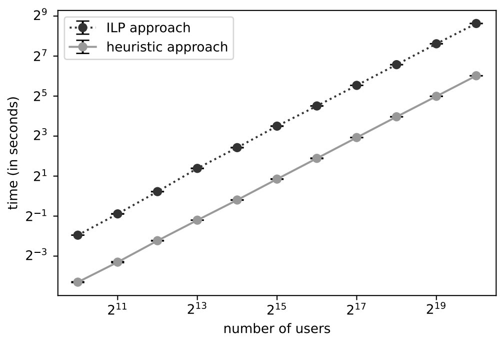
Question 3 (*What are the important factors that influence the runtime of the privacy planner?*)

To answer this question, we measure the runtime of individual components of the prototype. In addition, we perform a small ablation study to look at the influence of different types of queries.

In Figure 6.5 we present the runtime of individual components for both the ILP (Figure 6.5a) and heuristic approach (Figure 6.5b). In the ILP approach, adding constraints and running the optimization in Gurobi takes most of the total runtime (81% for $500K$ users). Interestingly, Gurobi requires longer for adding the con-



(a) linear scale



(b) loglog scale

Figure 6.4: The runtime of our ILP and heuristic approaches compared.

straints compared to optimizing the ILP. More than half of the total runtime of the ILP approach (56% for $500K$ users) comes only from adding constraints.

As we have already referred to earlier, Figure 6.3 shows the number of constraints, which scale linearly with the number of users. With $500K$ users, we have (on average) 2.9 million variables and 4.5 million constraints, and double these amounts for one million users.

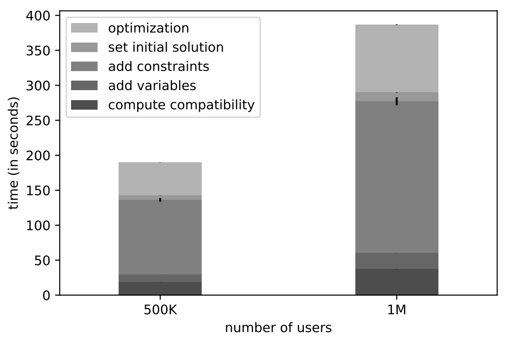
That adding constraints is a main bottleneck for the ILP approach has several important reasons:

1. We think our type of ILP problem is a comparatively easy one for Gurobi. We know that in the worst case, the optimization could scale exponentially. Seeing that the time needed for optimization is small, even for one million users, we do not think there is any exponential scaling of practical importance, reinforcing the idea of a not too difficult problem instance for Gurobi.

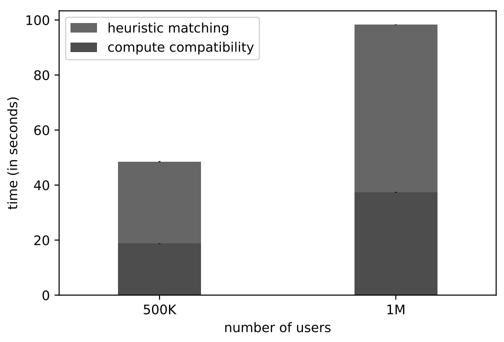
2. While the optimizer makes use of multiple threads, adding constraints is single-threaded. Combined with the fact that the z1d.2xlarge instances used for this evaluation feature eight vCPU's, it seems plausible that this significantly affected the relative runtime of optimization and adding constraints.

3. Gurobi already builds up some internal data structures when constraints (and variables) are added, increasing the runtime of adding constraints while favoring the optimization runtime.

In the heuristic approach, there are only two components. First, we compute the compatibility between privacy policies and (virtual) queries, and then we run the heuristic matching as described in Section 4.3. Figure 6.5b shows that, for $500K$ users, 39% of the total runtime is used for computing the compatibility, while



(a) ILP approach



(b) heuristic approach

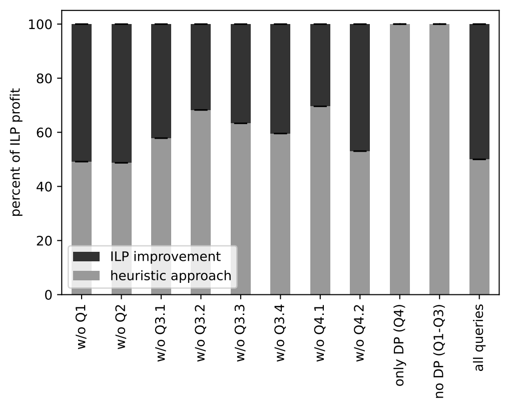
Figure 6.5: What the runtime proportions of different elements of the two approaches are. See Section 5 for an overview over the different components from a functional perspective.

the remaining 61% is used for greedily assigning users to queries. Since we are comparing every user and virtual query, it is no surprise that the compatibility part takes some time as well. Caching the compatibility between privacy policies (of which there are a lot less than users) and queries undoubtedly helps in this regard; otherwise, this part might even dominate runtime.

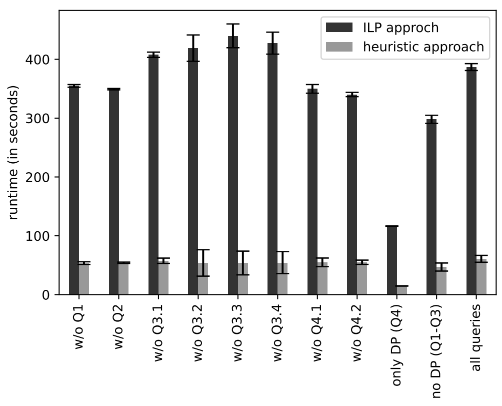
Overall, the measurements reveal nothing surprising for our heuristic approach, but for the ILP approach, it is clear that adding constraints to Gurobi is a significant bottleneck. This fact leaves room for optimizations in future work.

Query Ablation Study To study the impact of different types of queries (Table 3), we perform an ablation study. In Figure 6.6 we show the profit (Figure 6.6a) and runtime (Figure 6.6b) when we remove individual queries or a group of queries with similar properties (e.g., all queries with differential privacy). Removing Queries seems to have significant effects on profit, depending on which query was removed. When we have either exclusively Queries with DP or no Queries with DP, the heuristic algorithm can find (almost) optimal solutions, which are not further improved by our ILP approach. If we have only Queries with DP left, it is no big surprise that the problem changes significantly since we then only have two queries left (see Table 3). In addition, each user that could be assigned to $Q4.1$ can also be assigned to $Q4.2$ (for less profit). Such a setup is optimal for our heuristic approach, which explains why the ILP approach does not net any benefit here.

On the other hand, it is initially unclear why the heuristic approach finds near-optimal solutions if we only have queries without differential privacy. One factor



(a) profit with ILP as baseline



(b) Runtime of the two approaches

Figure 6.6: The impact of removing single or multiple queries from our evaluation problem instance.

might be that Queries 3.1 - 3.4 are non-overlapping, in the sense that a user which is part of query $3.i$ cannot be part of query $3.j$ for $i \neq j$. Query 1 also only overlaps with query 3.1. Only query 2 overlaps with all Queries $3.k$. Since the queries, therefore, almost "divide up" the available users without much overlap, the heuristic algorithm and the ILP approach do not have much choice where to assign users, which is why even the greedy heuristic approach gets near-optimal results.

The experimental results suggest that in more complex real-world deployments with more competition for users, the ILP approach could bring an essential advantage. However, this has to be investigated in more depth in future work.

In terms of runtime, removing queries has primarily predictable effects. That the runtime is drastically shorter if we only have queries with differential privacy is not surprising, again considering only two such queries in our example set of queries exist. However, there is one exception: Removing one of the queries $3.i$ seems to increase the runtime of the ILP approach. However, we know that adding more constraints can make an ILP easier to solve in some cases. As a result, it is hard to predict what effect removing some constraints (as is the case when removing query $3.i$) has on runtime for the ILP solver and our ILP approach as a whole.

Start with initial Solution In Gurobi, we can set an initial feasible solution as a starting point for the optimization. In the prototype, we use the assignment from the heuristic approach as a starting point for the ILP.

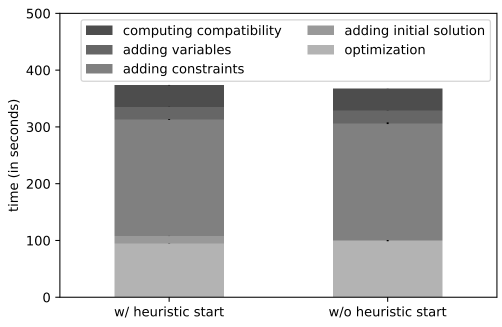


Figure 6.7: Runtime of our ILP approach where an initial solution obtained from the heuristic approach was set, compared to the runtime with no initial solution set, both with one million users.

According to the Gurobi documentation, ”it can be helpful for the modeler to provide a feasible solution along with the model itself”, for problem instances where ”the MIP solver is slow in finding an initial feasible solution” [16].

Recall that we run the ILP optimization until we find a solution within 5% to the optimal solution. Therefore, starting with a good initial solution promises to find a near-optimal solution more quickly, resulting in a better runtime.

In Figure 6.7, we compare the runtime with and without the heuristic start. We can see that the total runtime of the ILP approach is lower if the optimizations start with this heuristic solution due to the time needed to input the solution into Gurobi. Consequently, the experimental data suggest that receiving the heuristic solution does not improve the runtime of our heuristic approach.

However, starting with the heuristic solution is still helpful because it guarantees that the ILP solution is always at least as good as the heuristic solution. This situation can happen in rare circumstances since Gurobi stops after finding a solution at most 5% away from an optimal one, while the heuristic algorithm might find an optimal solution. Another way to tackle this could be to decrease tolerance for Gurobi, such that it only stops after finding a solution that is at most, e.g., $0.0001 = 0.1\%$ (default value) away from an optimal solution. However, that would likely increase runtime more, though with the added benefit of leading to better solutions when the heuristic solution is far from optimal.

In conclusion, we identified many vital factors that influence the runtime of the privacy planner. First, of course, the size of the problem (number of users and number of queries) is an essential factor. Second, the exact problem instance (e.g., which queries are specified) can have substantial impacts. Third, we identified that adding constraints is the main bottleneck for our ILP approach. Finally, forwarding the heuristic solution to Gurobi does not improve runtime in our experiments.

7 Conclusion

Recent trends towards more collection of sensitive data and a push for increased privacy protections are likely to continue for the foreseeable future. While a flurry of new privacy systems is being developed, new challenges are also arising.

In this thesis, we address the privacy management challenges that arise in user-centric privacy systems that deal with a heterogeneous set of privacy policies and a diverse set of queries by the provider. We present a solution that finds assignments of users to queries that conform to all applicable privacy policies. Either using our fast, heuristic approach or our integer linear program (ILP) approach, which can find near-optimal solutions (with a configurable bound) at the cost of increased runtime. By executing queries according to this assignment, data can then be transformed into a policy-compliant view, on which unrestricted analyses can be run.

We showed that both of these solutions are feasible in offline, batch-oriented settings, and that they scale reasonably well (linear scaling in runtime and memory regarding the number of users). While the ILP approach needs significantly more computing resources, we show that its solutions can significantly outperform the heuristic algorithm (almost double the profits for one million users). Still, in situations where there is little competition for users' data (i.e., most users are only compatible with a single query), the heuristic algorithm can achieve near-optimal solutions.

This thesis addresses a broad challenge that arises in any user-centric privacy system. Namely, how to manage a heterogeneous set of privacy policies such that the generated value is maximized. By maximizing the value generated from data, the cost in data utility from implementing more strict privacy protections decreases. As effects on data utility are a primary concern when implementing privacy measures, we argue that our approach can help to increase adoption of any user-centric privacy system making use of our approaches.

7.1 Future Work

While we addressed several challenges that come up concerning privacy management, many key issues remain. In the following, we state some of the most important future research questions related to the research in this thesis.

- *Implementing data transformations*: We outlined (see Figure 4.1) that the solutions found by our privacy planner can be used to compute a policy-compliant view of data. However, implementing such data transformations was outside the scope of our thesis. Towards this end, future research could either implement such data transformations directly or integrate our approaches into already existing end-to-end data privacy systems (such as Zeph [4]).

- *Mapping Privacy Policies*: As outlined in Section 4.2 (see Figure 4.2), we assume a system where users set valid privacy preferences across applications and that these are then translated to privacy policies by experts. However, we did not detail how such privacy preferences could look, what rules could govern the translation into privacy policies, and if such translation could be (partially) automatic. Research into such problems is essential not only to this thesis but for the whole field of data privacy. We argue that it is even a prerequisite for more widespread adoption of user-centric privacy systems.

References

[1] Apache kafka. URL: https://kafka.apache.org/ (visited on 04/05/2021).

[2] A. H. Assiciation. Target heart rates chart. URL: https://www.heart.org/en/healthy-living/fitness/fitness-basics/target-heart-rates (visited on 05/03/2021).

[3] E. Bagdasaryan, G. Berlstein, J. Waterman, E. Birrell, N. Foster, F. B. Schneider, and D. Estrin. Ancile: enhancing privacy for ubiquitous computing with use-based privacy. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society*, pages 111–124, 2019.

[4] L. Burkhalter, N. Küchler, A. Viand, H. Shafagh, and A. Hithnawi. Zeph: cryptographic enforcement of end-to-end data privacy. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 387–404, 2021.

[5] H. Corrigan-Gibbs and D. Boneh. Prio: private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 259–282, 2017.

[6] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. In volume 9, 2014.

[7] Electronic frontier foundation. URL: https://www.eff.org/ (visited on 07/31/2021).

[8] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[9] Facebook sued for 'losing control' of users' data, Feb. 9, 2021. URL: https://www.bbc.com/news/technology-55998588 (visited on 07/16/2021).

[10] Federal act on data protection. URL: https://www.fedlex.admin.ch/eli/cc/1993/1945_1945_1945/en (visited on 07/04/2021).

[11] Federal Statistical Office / Bundesamt für Statistik (BFS). Analyseregionen. URL: https://www.bfs.admin.ch/bfs/de/home/statistiken/querschnittsthemen/raeumliche-analysen/raeumliche-gliederungen/analyseregionen.html (visited on 06/12/2021).

[12] Federal Statistical Office / Bundesamt für Statistik (BFS). Ständige Wohnbevölkerung nach Altersklasse und Altersmasszahlen nach Kanton, definitive Jahresergebnisse, 2019. URL: https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/stand-entwicklung/alter-zivilstand-staatsangehoerigkeit.assetdetail.13707296.html (visited on 06/12/2021).

[13] Gdpr online. URL: https://gdpr.eu/ (visited on 07/04/2021).

[14] D. J. Glancy. The invention of the right to privacy. In *Arizona Law Review v. 21*, 1979.

[15] Google has gotten incredibly good at predicting traffic — here's how. URL: `https://www.businessinsider.com/how-google-maps-knows-about-traffic-2015-11?r=US&IR=T` (visited on 07/19/2021).

[16] Gurobi mip start. URL: `https://www.gurobi.com/documentation/9.1/examples/mip_starts.html` (visited on 08/19/2021).

[17] How aws uses automated reasoning to help you achieve security at scale. URL: `https://aws.amazon.com/blogs/security/protect-sensitive-data-in-the-cloud-with-automated-reasoning-zelkova/` (visited on 07/30/2021).

[18] G. Kellaris, S. Papadopoulos, X. Xiao, and D. Papadias. Differentially private event sequences over infinite streams. *Proc. VLDB Endow.*, 2014.

[19] M. Lécuyer, R. Spahn, K. Vodrahalli, R. Geambasu, and D. Hsu. Privacy accounting and quality control in the sage differentially private ml platform. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 181–195, 2019.

[20] T. Luo, M. Pan, P. Tholoniat, A. Cidon, R. Geambasu, and M. Lécuyer. Privacy budget scheduling. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 55–74, 2021.

[21] A. Marzoev, L. T. Araújo, M. Schwarzkopf, S. Yagati, E. Kohler, R. Morris, M. F. Kaashoek, and S. Madden. Towards multiverse databases. In HotOS '19, pages 88–95, 2019.

[22] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, 2009.

[23] A. Mehta, E. Elnikety, K. Harvey, D. Garg, and P. Druschel. Qapla: policy compliance for database-backed systems. In *26th USENIX Security Symposium (USENIX Security 17)*, 2017.

[24] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125, 2008.

[25] H. Nissenbaum. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press, 2010.

[26] Privacy on iphone — tracked — apple. URL: `https://www.youtube.com/watch?v=8w4qPUSG17Y` (visited on 07/16/2021).

[27] Privitar privacy platform overview. URL: `https://www.privitar.com/resources/privacy-platform-overview/` (visited on 03/15/2021).

[28] Python enum. URL: https://docs.python.org/3.8/library/enum.html (visited on 08/01/2021).

[29] Python lambda functions. URL: https://docs.python.org/3.8/tutorial/controlflow.html (visited on 08/01/2021).

[30] Python time. URL: https://docs.python.org/3/library/time.html (visited on 08/19/2021).

[31] Rényi differential privacy. *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017.

[32] Roundup of machine learning forecasts and market estimates, 2020. URL: https://www.forbes.com/sites/louiscolumbus/2020/01/19/roundup-of-machine-learning-forecasts-and-market-estimates-2020/ (visited on 07/19/2021).

[33] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression, 1998.

[34] Study: reading online privacy policies could cost $365 billion a year. URL: https://arstechnica.com/tech-policy/2008/10/study-reading-online-privacy-policies-could-cost-365-billion-a-year/ (visited on 07/31/2021).

[35] Y. Sun, L. Yin, L. Liu, and S. Xin. Toward inference attacks for k-anonymity. In *Personal and Ubiquitous Computing, Volume 18*, 2014.

[36] F. Wang, R. Ko, and J. Mickens. Riverbed: enforcing user-defined privacy constraints in distributed web services. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 615–630, 2019.

[37] S. D. Warren and L. D. Brandeis. Right to privacy. In *Harvard Law Review 4*, 1890.

[38] L. Waye. Privacy integrated data stream queries. In *Proceedings of the 2014 International Workshop on Privacy &; Security in Programming*, 2014.

[39] We read 150 privacy policies. they were an incomprehensible disaster. URL: https://www.nytimes.com/interactive/2019/06/12/opinion/facebook-google-privacy-policies.html (visited on 07/31/2021).

List of Definitions, Theorems, and Examples

List of Figures

List of Tables

A Appendix

A.1 Evaluation Details

In the following, we will detail the full example including all parameters that was used in Section 6: Evaluation to assess the performance of our prototype. We start by showing the data schema, privacy options and privacy policies that were part of this example. Then, we detail the set of queries and how users were generated.

Data Schema

EventStream1

- **name**: ”heartRate”
  type: $long$
  range: $[0, 240]$
  description: heart rate in beats per minute

UserLevelMetadata

- **name**: ”age”
  type: $enum$
  symbols: [”0-19”, ”20-39”, ”40-64”, ”65-79”, ”80+”]
  description: Age classes in years as used by the Federal Statistical Office [12]
- **name**: ”residentialRegion”
  type: $enum$
  symbols: [”Lake Geneva Region”, ”Espace Mittelland”, ”Northwestern Switzerland”, ”Zurich”, ”Eastern Switzerland”, ”Central Switzerland”, ”Ticino”]
  description: regions for analysis purposes in Switzerland as used by the Federal Statistical Office [11]

Privacy Options

- **name**: ”Private”
  chain: ”Deletion”
  transformations: [(”RedField”, {})]
  description: user-level differential privacy with $\epsilon = 2$ and $\delta = 0$, supporting the release of differentially private aggregations sum, count and variance. There is no minimum requirement of people, although if one does an aggregation with only a few people noise will dominate.

- **name**: ”Differential Privacy”
  chain: ”Private Sharing”
  transformations: $[$
  (”TimeRes”, $\{86400,$ [”sum”,”count”,”variance”]$\}$),
  (”PertDP”, $\{$”user-level”, $2, 0,$”laplace”$\}$)
  $]$
  description: user-level differential privacy with $\epsilon = 2$ and $\delta = 0$, supporting the release of differentially private aggregations sum, count and variance. There is no minimum requirement of people, although if one does an aggregation with only a few people noise will dominate.

- **name**: ”Heart Rate Buckets”
  chain: ”Private Sharing”
  transformations: $[$
  (”RangeRed”, $\{60, 100\}$),
  (”Bucket”, $[\{0, 5\}, \{5, 10\}, \{10, 20\}, \{20, 40\}, \{40, 60\}, \{100, 120\},$
  $\{120, 140\}, \{140, 160\}, \{160, 180\}, \{180, 200\}, \{200, 220\}, \{220, 240\}]$)
  $]$
  description: Bucketed Heart Rate values in beats per minute (bpm) without resting heart rate (ca. 60-100 bpm in adults according to the American Heart Association [2])

- **name**: ”Weekly Aggregates”
  chain: ”Private Sharing”
  transformations: $[$(”TimeRes”, 604800)$]$
  description: Only share weekly aggregates, with no pop requirements

- **name**: ”1000 People Hourly”
  chain: ”Private Sharing”
  transformations: $[$
  (”TimeRes”, $\{3600,$ [”sum”,”count”,”variance”]$\}$),
  (”PopRes”, $\{1000,$ [”sum”,”count”,”variance”]$\}$),
  $]$
  description: Aggregations sum, count and variance available in aggregation with 1000 ppl, with a min. time resolutions of one hour

- **name**: ”100 People Daily”
  chain: ”Private Sharing”
  transformations: [
  (”TimeRes”, $\{86400$, [”sum”,”count”,”variance”]$\})$,
  (”PopRes”, $\{100$, [”sum”,”count”,”variance”]$\})$,
  ]
  description: Aggregations sum, count and variance available in aggregation with 1000 ppl, with a min. time resolutions of one day
- **name**: ”Public”
  chain: ”Public”
  transformations: $[(UnmodRelease,)]$
  description: No restrictions on data usage

Privacy Policies

- **policy 1**
  EventStream1
  - **valueStreams**: ”heartRate”
    privacyOptions: [”Private”]
- **policy 2**
  EventStream1
  - **valueStreams**: ”heartRate”
    privacyOptions: [”Differential Privacy”]
- **policy 3**
  EventStream1
  - **valueStreams**: ”heartRate”
    privacyOptions: [”Differential Privacy”, ”1000 People Hourly”]
- **policy 4**
  EventStream1
  - **valueStreams**: ”heartRate”
    privacyOptions: [”Differential Privacy”, ”100 People Daily”]
- **policy 5**
  EventStream1
  - **valueStreams**: ”heartRate”
    privacyOptions: [”Differential Privacy”, ”100 People Daily”, ”1000 People Hourly”, ”Weekly Aggregates”]

- **policy 6**
  EventStream1
  - **valueStreams**: ”heartRate”
    privacyOptions: [”Differential Privacy”, ”100 People Daily”, ”1000 People Hourly”, ”Weekly Aggregates”, ”Heart Rate Buckets”]
- **policy 7**
  EventStream1
  - **valueStreams**: ”heartRate”
    privacyOptions: [”Public”]

For all policies we also assume:

UserLevelMetadata

- **valueStreams**: ”age”
  privacyOptions: [”Public”]
- **valueStreams**: ”residentialRegion”
  privacyOptions: [”Public”]

Queries

- **name**: ”Zurich over 65 bucketed”
  inputSchema: ”heartRate”
  chain: ”Private Sharing”
  transformations: [
  (”RangeRed”, $\{60, 100\}$),
  (”Bucket”, $[\{0, 5\}, \{5, 10\}, \{10, 20\}, \{20, 40\}, \{40, 60\}, \{100, 120\}, \{120, 140\}, \{140, 160\}, \{160, 180\}, \{180, 200\}, \{200, 220\}, \{220, 240\}]$)
  conditions: residentialRegion$==$”Zurich” $\&\&$ age$>= 65$
  numberAndProfit $[(100, 100), (200, 50), (200000, 10)]$
- **name**: ”Ticino bucketed”
  inputSchema: ”heartRate”
  chain: ”Private Sharing”
  transformations: [
  (”RangeRed”, $\{60, 100\}$),
  (”Bucket”, $[\{0, 5\}, \{5, 10\}, \{10, 20\}, \{20, 40\}, \{40, 60\}, \{100, 120\}, \{120, 140\}, \{140, 160\}, \{160, 180\}, \{180, 200\}, \{200, 220\}, \{220, 240\}]$)
  conditions: residentialRegion$==$”Ticino”
  numberAndProfit $[(100, 100), (200, 50), (200000, 10)]$

- **name**: ”Hourly Aggregated heart rates 65 or older”
  inputSchema: ”heartRate”
  chain: ”Private Sharing”
  transformations: $[($”TimeRes”, $\{3600, [$”sum”,”count”,”variance”$]\}),$ $($”PopRes”, $\{100, [$”sum”,”count”,”variance”$]\})]$
  conditions: age $>= 65$
  numberAndProfit: $[(10, 9000), (200, 5000), (200000, 1000)]$

- **name**: ”Hourly Aggregated heart rates between 40-64”
  inputSchema: ”heartRate”
  chain: ”Private Sharing”
  transformations: $[($”TimeRes”, $\{3600, [$”sum”,”count”,”variance”$]\}),$ $($”PopRes”, $\{100, [$”sum”,”count”,”variance”$]\})]$
  conditions: age $>= 40$ && age $<= 64$
  numberAndProfit: $[(10, 9000), (200, 5000), (200000, 1000)]$

- **name**: ”Hourly Aggregated heart rates between 20-39”
  inputSchema: ”heartRate”
  chain: ”Private Sharing”
  transformations: $[($”TimeRes”, $\{3600, [$”sum”,”count”,”variance”$]\}),$ $($”PopRes”, $\{100, [$”sum”,”count”,”variance”$]\})]$
  conditions: age $>= 20$ && age $<= 39$
  numberAndProfit: $[(10, 9000), (200, 5000), (200000, 1000)]$

- **name**: ”Hourly Aggregated heart rates 19 or younger”
  inputSchema: ”heartRate”
  chain: ”Private Sharing”
  transformations: $[($”TimeRes”, $\{3600, [$”sum”,”count”,”variance”$]\}),$ $($”PopRes”, $\{100, [$”sum”,”count”,”variance”$]\})]$
  conditions: age $<= 19$
  numberAndProfit: $[(10, 9000), (200, 5000), (200000, 1000)]$

- **name**: ”Daily Differentially Private”
  inputSchema: ”heartRate”
  chain: ”Private Sharing”
  transformations: $[$
  (”TimeRes”, $\{86400$, [”sum”,”count”,”variance”]$\}$),
  (”PopRes”, $\{100$, [”sum”,”count”,”variance”]$\}$),
  (”PertDP”, $\{$”user-level”, $0.01, 0, 1, 0$,”laplace”$\}$)
  $]$
  numberAndProfit: $[(100000, 3000)]$

- **name**: ”Weekly Differentially Private”
  inputSchema: ”heartRate”
  chain: ”Private Sharing”
  transformations: $[$
  (”TimeRes”, $\{604800$, [”sum”,”count”,”variance”]$\}$),
  (”PopRes”, $\{100$, [”sum”,”count”,”variance”]$\}$),
  (”PertDP”, $\{$”user-level”, $0.01, 0, 0.6, 0$,”laplace”$\}$)
  $]$
  numberAndProfit: $[(100000, 1000)]$

User Distribution

In this section, we detail how users where assigned. As mentioned in Section 6.1, users are assigned privacy policies uniformly at random from the ones available in Section A.1. Metadata is assigned independently from privacy policies, according to the distribution detailed in Table 4

Region	0-19	20-39	40-64	65-79	80+
Lake Geneva Region	350184	451997	567605	202793	82172
Espace Mittelland	373779	481200	656842	270362	104401
North-western Switzer-land	228236	301758	415191	161886	64086
Zurich	304095	441221	531563	186909	75487
Eastern Switzer-land	234375	307628	413727	166169	61914
Central Switzer-land	163404	213605	293050	108201	40702
Ticino	63125	77117	130531	55810	24908

Table 4: Distribution of age and residential region in Switzerland according to data from the Swiss Federal Statistical Office [12]