AFRL-RI-RS-TR-2021-123

NOVEL TOOLS FOR ANALYZING PRIVACY LEAKAGES (NAPLES)

CYBERNETICA AS

SEPTEMBER 2021

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA).

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Defense Advanced Research Projects Agency (DARPA) Public Release Center and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2021-123 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
CARL R. THOMAS
Work Unit Manager

/ S /
GREGORY J. HADYNSKY
Assistant Technical Advisor,
Computing & Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
SEPTEMBER 2021	FINAL TECHNICAL REPORT	AUG 2015 – JAN 2021

4. TITLE AND SUBTITLE

NOVEL TOOLS FOR ANALYZING PRIVACY LEAKAGES (NAPLES)

5a. CONTRACT NUMBER
FA8750-16-C-0011

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
62303E

6. AUTHOR(S)
Sara Belluccini, Dan Bogdanov, Mykhailo Dorokhov, Marlon Dumas, Luciano García-Bañuelos, Joosep Jääger, Marko Jõemets, Peeter Laud, Raimundas Matuleviius, Markko Merzin, Henry Mägi, Alisa Pankova, Martin Pettai, Pille Pullonen, Jaak Randmets, Jaak Ristioja, Angela Sahk, Andrey Sergeev, Jake Tom, Maria Toomsalu, Aivo Toots, Reedik Tuuling, Sander Valvas, and Maksym Yerokhin

5d. PROJECT NUMBER
BRAN

5e. TASK NUMBER
CV

5f. WORK UNIT NUMBER
BT

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Cybernetica AS,
Mäealuse 2/1,
12618 Tallinn, Estonia

University of Tartu, Institute of Computer Science,
Narva mnt. 18,
51009 Tartu, Estonia

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

DARPA
675 North Randolph Street
Arlington, VA 22203-2114

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2021-123

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. DARPA DISTAR CASE # 35039
Date Cleared: 14 September 2021

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We have proposed a large number of modeling and analysis tools for characterizing the privacy properties of business processes. Our modeling annotations and analyses, building upon the Business Process Modeling Notation (BPMN), fall into three major levels of abstraction. The first level allows us to see which input data sources are directly or indirectly disclosed to each party in the process. The second level shows under what conditions each disclosure occurs, and which specific attributes are disclosed. The third level allows the measurement of the extent to which the disclosure reveals information about individual items. Most analyses are integrated into our Pleak tool, which gives them a unified interface.

15. SUBJECT TERMS
Privacy Analysis, Business Process Model and Notation, Differential Privacy, Privacy Enhancing Technologies, PE-BPMN, PLEAK

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**CARL R. THOMAS**
U	U	U	UU	316	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 SUMMARY

In NAPLES project, we proposed qualitative and quantitative definitions of privacy suitable for business processes, developed static analyses allowing to check whether a process satisfies a privacy definition, and integrated the modeling tools and analyses into a single coherent tool. The tool was further used to model and analyze processes by other performers of the Brandeis program, which the NAPLES project was a part of.

Our proposed privacy definitions and analyses fall into three different categories. In the first category, we have methods that add expressive annotations to the collaborative business process models, telling which privacy-enhancing technologies are used by them and how they connect with each other. The analyses accompanying the modeling methods find out, whether a dataset may become known to some participant of the process, and if it may, then is it perhaps still protected by some privacy-enhancing technology. The second category of analyses take into account the internal structure of the datasets, and make use of precise descriptions of the operations that the tasks in the business process apply to the datasets. The analyses report, which component of which input dataset may influence which component of which output dataset, under which condition the influence is possible, and what kind of processing must happen to the influencing information. These influences may be compared against the policies the participants may have on their data. The third category of modeling tools and analyses allows the leaks to be characterized quantitatively. Building upon the notions of differential privacy and guessing advantage, we develop a rich language for stating quantitative privacy policies. Again considering the operations performed by the tasks of the business process, our analyses give a numeric description of the amount of information that reaches the output datasets from the inputs.

In this report, we give a detailed description of our modeling and analysis techniques, as well as describe the experiments and evaluations we have done with them either on our own, or in cooperation with other performers of DARPA's Brandeis program. We also describe the Pleak tool, which gave a unified interface to most of our analyzers. Finally, we describe a transition activity related to cyber threat information sharing, which involved both the modeling of sharing processes, as well as their implementation. In the implementation, we chose to apply secure multiparty computation; the effect of this privacy-enhancing technology was analyzed with the help of Pleak.

2 INTRODUCTION

This is the final report of the "Novel tools for Analyzing Privacy LEakageS (NAPLES)" project, which ran from October 2015 till January 2021 with the funding from DARPA's "Brandeis" program [1]. The program sought to develop the technical means to protect the private and proprietary information of individuals and enterprises, thereby enabling safe and predictable sharing of data in which privacy is preserved. In this report, we present a detailed account of the results achieved during the project.

2.1 Initial Goals of The Project

In NAPLES, we strove to create methods for analyzing and improving the privacy of enterprise business processes (BP), and to package these methods as a tool. The tool was intended to be used by BP analysts, developers and maintainers in order to understand the privacy implications of the business processes used by their organizations. In order to come up with such methods, and to build such tool, we needed to research, design and implement the following results and artifacts:

Framework of privacy claims We intended to propose a quantitative definition of privacy that is at the same time composable, easily explainable, and allows its granularity to be tuned to the particular BP and BP components we are analyzing.

Library of Privacy Enhancing Technologies (PETs) This library was intended to contain multiple methods to enhance activities or other components of a business process with secure computing and information release mechanisms involving data reorganization, sanitation or processing.

Program privacy analysis Given the operational description of a BP component in the form of a program with clear semantics, we planned to develop techniques deriving a characterization of this component in terms of its privacy behavior, or a conservative approximation of it.

Business process privacy analysis Given a business process described in the Business Process Model and Notation (BPMN), and given a characterization of the privacy behaviors of the components of the BP, this analysis will characterize the privacy behavior of the entire BP by exploiting the composability of the proposed privacy definition. Furthermore, given the components available in the PETs library and given a privacy policy, we intended to come up with ways to propose semantics-preserving enhancements of the BP to achieve the policy in question. We also intended to quantitatively characterize the (loss of) accuracy of BP components when enhanced with different PETs and take into account trade-offs between accuracy and privacy when optimizing business processes.

Toolbox We intended to package the analyses and optimizations in a graphical tool built on top of an existing open-source BPMN tool used by practitioners. We were eyeing bpmn.io as the basis of our toolset.

2.2 Modeling Collaborative Business Processes

A business process is a collection of tasks and activities consisting of employees, materials, machines, systems, and methods that are being structured in such a way as to design, create, and deliver a product or a service to the consumer [1].

[1] DARPA Brandeis—`http://www.darpa.mil/program/brandeis`

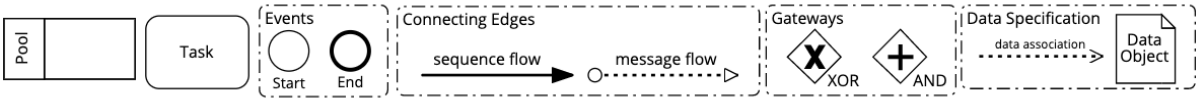


Figure 1: Elements of the BPMN Notation

The product, its raw materials, and the side-effects of its production may also be digital, raising questions about the privacy properties of the processes.

A suitable notation is needed to introduce rigor into discussions about business processes. *Business Process Model and Notation* (BPMN) [2] has emerged as the standard operational description of business processes. BPMN was originally developed to provide a notation that was easily understandable by all business users, from technical analysts implementing an information system to business analysts to business users who manage the processes. The BPMN notation allows one to capture processes performed within an organization and across organizations. The latter type of process is called a *collaborative process* and is represented by a *BPMN collaboration diagram* (or *BPMN collaboration* for short). A BPMN collaboration consists of a set of processes, each performed by an independent *party* (e.g., buyer and seller). These processes are executed in parallel and synchronize via message exchanges (dashed arcs). Each process in a BPMN collaboration is captured as a separate pool (denoted as a rectangle). A process consists of tasks (rounded rectangles), events (circles) and gateways (diamonds). A task represents a logical unit of work. An event represents something triggered by the environment (e.g., a message). A gateway is used to capture a choice (XOR gateways, marked by a "×") or the parallel execution or synchronization of multiple branches (AND gateways, marked with a "+"). These three types of elements (tasks, events, gateways) are connected via sequence flows (directed arcs). A sequence flow indicates that the source element must be executed before the target element. To capture data manipulation, each task may be associated (via directed dotted arcs) to one or more input or output data objects. The intended meaning is that when the task is executed, it reads the current state of each input object, and when it completes it writes into the output data objects. These concepts are summarized in Fig. 1.

Figure 2 shows a very simple business process at a telecommunications (telco) services provider. The telco provider is represented by a pool. There is a separate pool below it, corresponding to a contractor hired by the telco to provide services. To provide its services, the contractor needs to access weekly "service summary reports" produced by the telco. Inside the telco's pool, there are two roles represented by the lanes labeled "Data Analyst 1" and "Data Analyst 2". The process starts when a new summary report is created (cf. the start event labeled "summary report required"). First, Data Analyst 1 performs a (user) task wherein a set of call records are accessed in order to prepare a call summary table. This collection of call records (represented by a "collection" data object) may contain sensitive data. If no privacy protection mechanism is in place, then the data analyst sees all the call records. As a result of this task, a "Call summary table" is produced. Next, a script task is run that combines this "Call summary table" with another collection of "Data connection records", in order to produce a "Combined report". The combined report is checked by a Data Analyst 2, who may apply some modifications. The process ends with an event denoting the fact that the combined report is sent out to the contractor.

2.3 NAPLES Toolbox

In NAPLES, we built Pleak — `https://pleak.io` — the Privacy LEAKage analysis tool. It integrates the implementations of our modeling and analysis techniques into a graphical modeling and analysis environment that is familiar to business process analysts. Pleak takes as input a description of a business process in BPMN and allows users (analysts, developers) to define privacy properties of the BP components. Given this input, the tool produces a report indicating how much information from its different input data the BP leaks to each user and external application involved in the process.

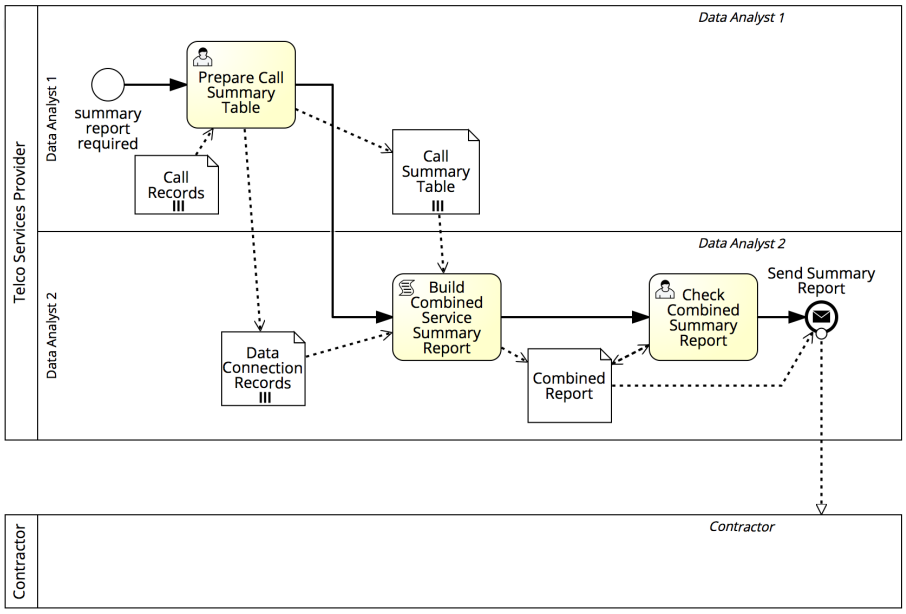


Figure 2: Example of a BPMN Process Model with Flows and Tasks

Pleak allows the models and analyses to be shared between different analysts. We considered Pleak to have two main usage scenarios.

Privacy audit of an existing system. In this scenario, the **stakeholders** are *System owner*–has built or otherwise acquired the *System*, *Customer*–the party who is interested in the audit results, can be the same party as the *System owner*, *Analyst*–the party undertaking the risk analysis or privacy audit. They would use Pleak as follows:

1. The *Customer* contracts the *Analyst* to analyze the *System* owned by the *System owner*.
2. The *Analyst* acquires details about the *System* from the *System owner* thorough documents, interviews or any other means.
3. The *Analyst* uses Pleak tools to model the *System*.
4. After modeling, the *Analyst* uses Pleak analytics and tools to answer the following questions.
   4.1 Which stakeholders of the *System* will learn which private data elements?
   4.2 To what extent wrt differential privacy does the leakage disclose private data?
5. The *Analyst* can specify undesired leakages and ask Pleak to suggest Privacy Enhancing Technologies to reduce or remove those leakages.
6. The *Analyst* collects all finds and presents them to the *Customer*.

Improving the privacy guarantees of a new IT system. In this scenario, the **stakeholders** are *Customer*–has commissioned the building of the *System*, *System developer*–the party who is building the system and wants to ensure that private data is processed, *Analyst*–member of the *System developer* team who is responsible for privacy guarantees of the system They would use Pleak as follows:

1. The *Customer* contracts the *System developer* to build the *System*.
2. The *System developer* assigns the *Analyst* to support the development team with the privacy guarantees of the *System*.
3. The *Analyst* collaborates with the *System* developer team to jointly design the *System*.
4. The *Analyst* uses Pleak tools to model the *System*'s design.
5. After modeling, the *Analyst* uses Pleak analytics and tools to answer the following questions.

5.1 Which stakeholders of the *System* would learn which private data elements?

5.2 To what extent wrt differential privacy would the leakage disclose private data?

6. The *Analyst* can specify undesired leakages and ask Pleak to suggest Privacy Enhancing Technologies to reduce or remove those leakages.
7. The *Analyst* collects all finds and works with the *System developer* team to improve the design.
8. The *Analyst* repeats the process with the improved system design until the desired privacy guarantees have been achieved.

We believe that a single tool integrating different analyzers, facilitating cooperation between analysts, and providing a unified user interface to analyses and modeling tools has been immensely helpful for its users among the different performers of Brandeis, and is currently helping the take-up of the results of NAPLES in follow-up projects, products, and services. We discuss the architecture of Pleak in Sec. 3.4 and describe its usage in Sec. 4.4.

2.4 Three Kinds of Analyses

In NAPLES, we have built a large number of privacy analyses and accompanying modeling tools supporting business processes of different kinds and different complexity. We find that there is a natural taxonomy of these analyses. Namely, considering the information we have about the data objects in a business process, and about the opera-tions applied to them, the analyses fall into the following three categories.

Boolean analyses. These analyses consider each dataset as a whole, and report whether or not a given (intermediate or final) output of a process may reveal information about a given input. These analyses request information about the nature of the tasks and the flows in the process, and use it to refine the nature of detected dependencies. The main kind of information that the analysis uses, concerns the usage of privacy-enhancing technologies at the tasks and the flows.

Qualitative analyses. These analyses explain how the outputs depend on inputs. If the data objects have structure, then the analysis result may describe, which components of the outputs depend on which components of the inputs. The explanation may state, that the dependency is present only if some predicate evaluates to true. It may also state that the dependency is there, but only through some sanitization function. In order to deliver these results, the analysis has to know how the tasks manipulate the data. Information about the usage of privacy-enhancing technologies can also be useful for the analyses.

Quantitative analyses. These analyses characterize the amount of information that flows from a particular input to a particular output. We have considered a number of different measures, according to which the analysis is done, mostly related to differential privacy and guessing advantage. In order to provide such results, these analyses also need to know how the components of the process transform the data. Contrary to the previous kinds, these analyses may also derive useful inferences from the actual values of input data objects, onto which the business process is applied.

Table 1 summarizes the different analyzers we have integrated into Pleak. All of them, as well as the theory behind their construction, are covered in further sections of this report.

2.5 Secure Multiparty Computation in NAPLES

During the final year of NAPLES, the scope of the project was extended to use Pleak[2] and Sharemind MPC[3,4] technologies as a part of an ongoing project. This served two purposes:

[2]Pleak web page: `https://Pleak.io/home`

[3]Sharemind web page: `https://sharemind.cyber.ee/`

[4]Sharemind product explainer: `https://www.youtube.com/watch?v=AVV35W-dehc`

Table 1: Summary of Analyzers

	Boolean		Qualitative		Quantitative		
	Simple and Extended Disclosure	**Leakage Detection**	**BPMN Leaks-When**	**SQL Leaks-When**	**Global Sensitivity**	**Combined Sensitivity**	**Guessing Advantage**
Model Type	Collaboration (multiple pools allowed)				Data processing workflow (single process)		Data processing workflow or Collaboration
Model restrictions	Only meaningful for multiple pools	Reasonable for models with branching	One start event only[a]	No branching		No branching, Final query has a numeric output (aggregation query)	
PETs support	All stereotypes	Secret sharing, Encryption	Encryption, Secure Channel	Partial support[b]	Differential privacy		
Script language	Not used		Pseudocode	Postgre SQL			
Input data	Not used					Required	
Policy	Not used			Data sharing policy	Not used		Guessing range
Attacker knowledge	Not used	Private data reconstruction	Not used			Attacker's prior knowledge about the data	

[a] over all the pools
[b] By integrating with the extended disclosure report

- Verify the usability of these technologies and learn what additional improvements may be required to increase the usefulness and usability of the technologies.
- Use Pleak and Sharemind capabilities in a real-life project.

The real-life project in for transitioning Pleak and Sharemind is the Cyber PA (Project Agreement) project[5].

In general, it was decided to

- Test information sharing and processing in a privacy-preserving environment where the information shared is not seen to the stakeholders, the processing of data is done in an encrypted form and the output of the processing contains valuable information for all contributors without revealing confidential information. The chosen privacy-preserving technology was the aforementioned Sharemind MPC.
- To use Pleak in order to model the processes of the Sharmind MPC prototype and additionally, use the Pleak tool, in order to model the preliminary Cyber PA project processes.

2.5.1 Cyber Project Agreement (PA). The main goal of the Cyber PA project is to enable cyber threat information sharing between defense forces of allied nations. As no one is alone in cyberspace and cyber threats are becoming more frequent, complex, destructive and coercive, information sharing and creation of joint cyber situational awareness is becoming increasingly important.

In order to find and notify partners of cyber threats one needs to be able to identify if and when such an event has taken place. With the ever-increasing amount of (system) logs being produced daily it is no longer feasible for a single individual / organization to be able to fully process and analyze all

[5] Cyber Project agreement press release: `https://cyber.ee/news/2020/01-14/`

log entries to detect anomalies. Attacks may also be devised in a way that no single entity can detect anomalies based solely on their individual logs. Furthermore, one can only find what they know to look for. Meaning for successful threat identification from logs, event correlation methods need to be used to bring together logs and threat identification markers from both government and public sector. Even though the key focus is to initiate cyber threat information sharing between nations it would be highly beneficial for national security if logs from critical organizations, like telecom provides, can also be used for correlation.

There is good reason why cyber threat information sharing and joint log correlation are not yet extensively used. The information contained within logs and the markers to identify potentially unwanted events are sensitive by nature. They may include personal or confidential information that cannot be shared due to legal restrictions or in order to protect the organization / nation. Also, nations may not wish to share with each other what sort of threats they are monitoring for. Security and privacy / confidentiality are highly critical in order to enable such sharing and log correlation.

2.5.2 Sharemind Multiparty Computation (MPC). Sharemind (illustrated in Figure 3) is a privacy-preserving database and analysis system that allows you to combine confidential data from several sources and analyze it without ever seeing the data itself. It also allows data owners and interested third parties to enforce data usage policies.

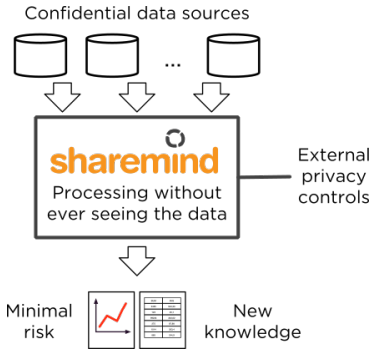


Figure 3: Sharemind Concept

Sharemind MPC is a practical implementation of secure multiparty computation technology with the emphasis on performance and ease of use. Three independent Sharemind hosts run the Sharemind MPC software. Data Owners share their data between the three Sharemind Hosts, resulting in a distributed database. Analysts can query this database, but only with the approval of all Hosts who verify that the query conforms to the data usage policy.

Sharemind MPC supports several different MPC schemes called *protection domains*, with the *shared3p* protection domain being the most advanced one. It stands for 3-out-of-3 secret sharing with passive security and uses additive secret sharing scheme, where a secret value s is secret shared as follows:

$$\begin{aligned} s_1 &\leftarrow \mathit{random}(), \\ s_2 &\leftarrow \mathit{random}(), \\ s_3 &\leftarrow s - s_1 - s_2, \end{aligned} \tag{1}$$

such that $s = s_1 + s_2 + s_3$. All these computations are done *modulo* the corresponding data type size, e.g. *modulo* 2^{64} for 64-bit (unsigned) integers. Note that this modulo computation happens automatically for primitive data types like *(u)int8, (u)int16, (u)int32* and *(u)int64*. More complex data types (e.g. floating-point numbers) use structures of primitive data types.

As an example consider Figure 4, suppose we have two Data Owners with their secret values – their ages 25 and 33 – and a third party – an Analyst – who wants to learn the sum of ages.

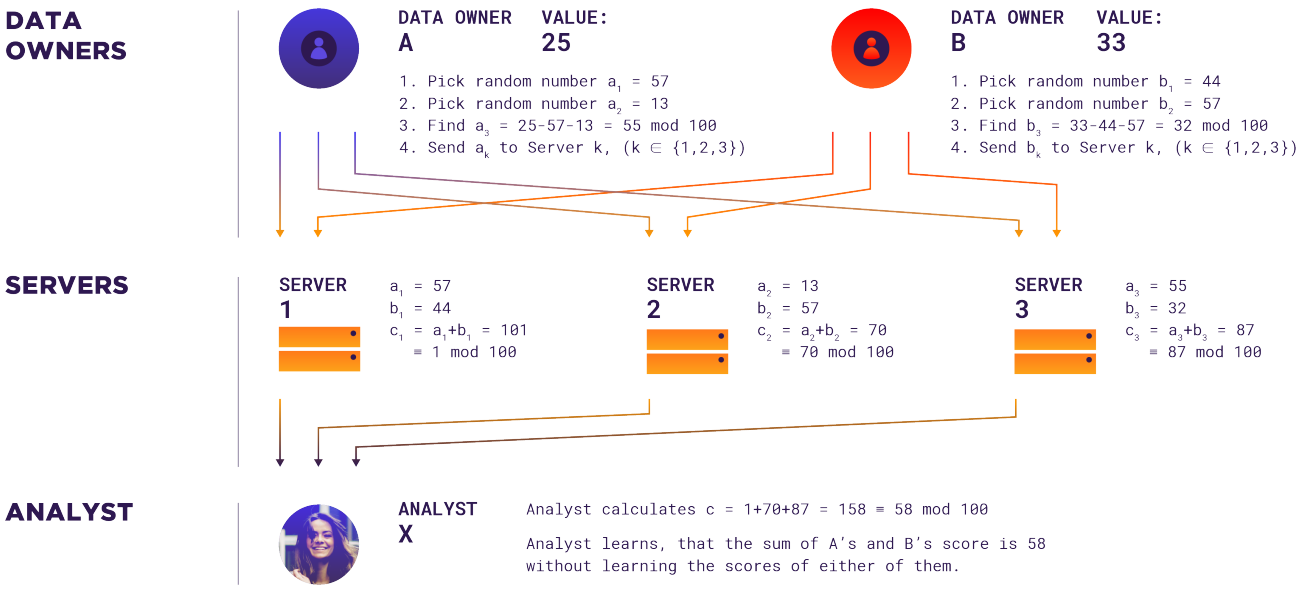


Figure 4: Secret Sharing Example Calculation

In additive secret sharing, to secret share her age, the first user picks two random numbers modulo 100. Let these be 57 and 13. The third share is computed by subtracting the random values from the secret value. 25 minus 57 minus 13 is -45, but modulo 100 it is the same as 55. The Data Owner now distributes these three shares – 57, 13 and 55 – between the three Sharemind hosts over secure authenticated and encrypted communication channels. None of the shares alone, nor any pair of these shares give any information about the original secret value. However, note that by adding up all the three shares, it is possible to reconstruct the original value. The second Data Owner acts similarly, he first generates two random numbers – 44 and 57 – and then computes the third share by subtracting these from the secret value 33. He distributes the shares among the Sharemind hosts. With additive secret sharing, addition is a local operation so each Sharemind host can just locally sum together their shares from the Data Owners. The first Sharemind host computes 57 plus 44 which is 101 but that is equal to 1 modulo 100. And so on. Next, each Sharemind host sends its partial result to the Analyst.

Finally, the Analyst reconstructs the computation result by adding up the individual received shares. 1 plus 70 plus 87 is equal to 158 that is equal to 58 modulo 100. And 58 is exactly 25 plus 33! Analyst has learned the sum of two ages without him learning their individual secret values. Sharemind hosts have not learned anything about the secret values not the computation result.

For even more information on Sharemind MPC please reference the Sharemind technical overview[6] and Developer Zone[7].

2.6 Collaborative Work

Being a part of the Brandeis program, we had a number of great collaborations with other performers. The collaborations took place inside the Collaborative Research Teams (CRT), set up at the beginning of the program. Our contributions to the CRTs have shaped the ways of how the technologies developed by different projects are integrated. The collaborations, which we detail in Sec.4.5, were an explicit goal of NAPLES, and we consider them significant results of the project.

We are grateful to DARPA and AFRL for the opportunity to perform this work, and to other performers of Brandeis for the fruitful collaborations.

[6]Sharemind Privacy Ecosystem Technical Overview: `https://repo.cyber.ee/sharemind/www/files/technology/sharemind-technical-overview.pdf`

[7]Sharemind Developer Zone: `https://docs.sharemind.cyber.ee/2019.03/prologue`

3 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 PE-BPMN

Privacy-Enhanced BPMN (PE-BPMN) notation is our extension of the BPMN modeling language to capture details of privacy enhancing technologies. PE-BPMN was specified in [3, 4] and this section gives an overview of the language extension as well as its implementation in Pleak.

3.1.1 Classification of Privacy Enhancing Technologies. We base our PET description on the PET classification from [5], our look on it is given in Table 2 as this generalisation is more useful for the privacy analysis. We identify different goals that usage of PET can have and group the technologies from [5] based on their goals. Note that sometimes the same PET could be used to obtain many goals. The overview of Table 2 lists some example technologies, the main body of the PET library considers the technologies used in the Brandeis program.

Table 2: Classification of Privacy Enhancing Technologies

Goal	Target	Examples of technology
Communication Protection	Secure	Client-Server encryption, TLS, IPSec, End-to-End encryption, PGP, OTR
	Anonymous	Proxies and VPN, onion routing, mix-networks, broadcast
Data Protection	Integrity	Message authentication codes, signatures
	Confidentiality	Encryption, secret sharing
Entity Authentication	Identity based	Username and password, single-sign-on
	Attribute based	Credential used only once, zero-knowledge proofs
Privacy Aware Computation	Confidential inputs	Homomorphic encryption, secure multiparty computation, private information retrieval
	Privacy adding	Differential privacy, k-anonymity, cell suppression, noise addition, aggregation, anonymization, dimensionality reduction
Human-Data Interaction	Data usage transparency	Information flow detection, logging, declarations about information usage
	Intervenability	Information granularity adjustment, access control

Secure communication means that the encrypted contents can travel between participants without external parties seeing or modifying them, e.g. providing both confidentiality and integrity. In *anonymous communication* the interacting parties can not be deduced by an observer. The general goal achieved in both cases is *communication protection*.

Data protection ensures integrity and confidentiality of the data either in transit or in storage. Hence, protected data can not be accessed or modified by non-authorized parties. Confidential data may allow confidentiality preserving computations that produce new confidential data.

Entity Authentication is a procedure for proving that user corresponds to the claimed attributes. Identity authentication requires some identity provider to verify all accesses (e.g., based on a fixed account). Attribute based methods deal with proving one's membership to some group, without identifying herself.

Privacy-Aware Computations focus on the utility of private data. Computations on confidential inputs allow one to securely process various operations without removing the protection mechanisms. We assume the ideal case where such computations fully protect the privacy of the inputs (up to the potentially public output). Privacy adding computations can add a layer of privacy to their outputs instead of fully protecting the inputs.

Human-Data Interaction is a field that combines technical means and policies with user experience. In essence, the users allowing some processing of their data should be knowledgeable about how and why their data is used. In addition, they may be able to regulate the data processing. These inputs are interleaved with security technologies that enforce the choices. Essentially, Pleak and all the analysis described in this report enable more transparent data usage in processes that are modeled and analyzed.

Another recent systematic comparison of properties of PETs is given in [6] that could be used to enhance the description of PETs and to build a decision tree. Our taxonomy combines the *aim*, *data* and *aspect* ideas of [6]. However, [6] considers more technical aspects that would be useful to consider also in the PETs data model. In addition, a different version of modeling privacy in data flows is considered in [7] to better discuss General Data Protection Regulation (GDPR) impacts for an organization.

3.1.2 Privacy-Enhanced BPMN. Fig. 5 presents extensions of BPMN abstract syntax [2] with the PET concepts.

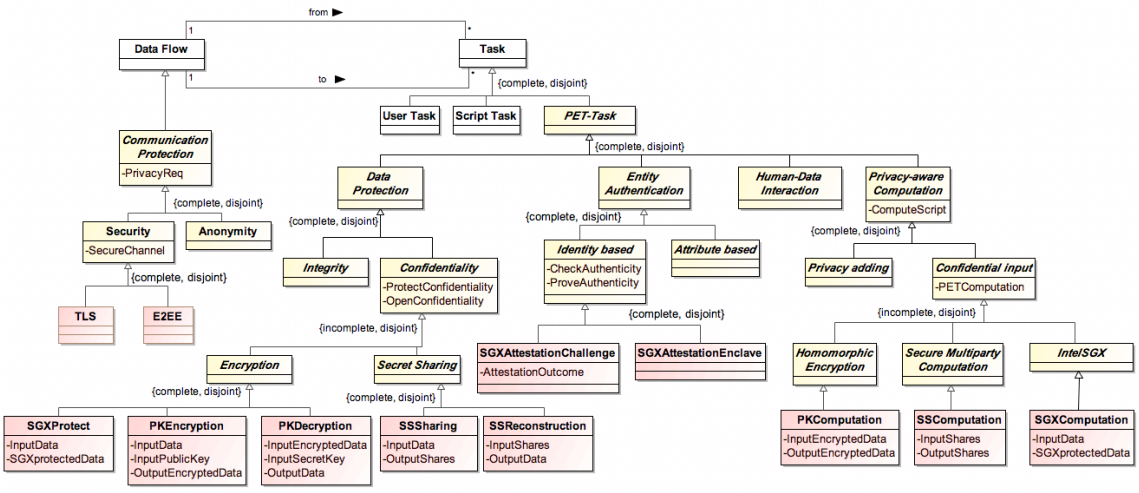


Figure 5: Extension of the BPMN Abstract Syntax

The BPMN Data Flow is extended with Communication Protection. In common secure channels, the message is hidden and can not be modified during transit. Thus, secure channels are straightforward to model in the sense that the communication and privacy risks occur between different pools. We introduce SecureChannel as a specialization of Communication Protection.

Most privacy related technologies result in specific tasks, thus BPMN Task is extended with abstract *PET-Task*. Fig. 5 illustrates four specializations of PET-Task: Data Protection, Entity Authentication, Privacy-Aware Computation and Human-Data Interaction. The figure also illustrates examples of syntax that we are using for concrete PETs.

Fig. 5 is not complete as other privacy technologies can be added from Table 2. However, it gives an example to both single task and multi-task technologies as well as secure communication, making extensions straightforward. We will consider concrete technologies in Sec. 3.1.3.

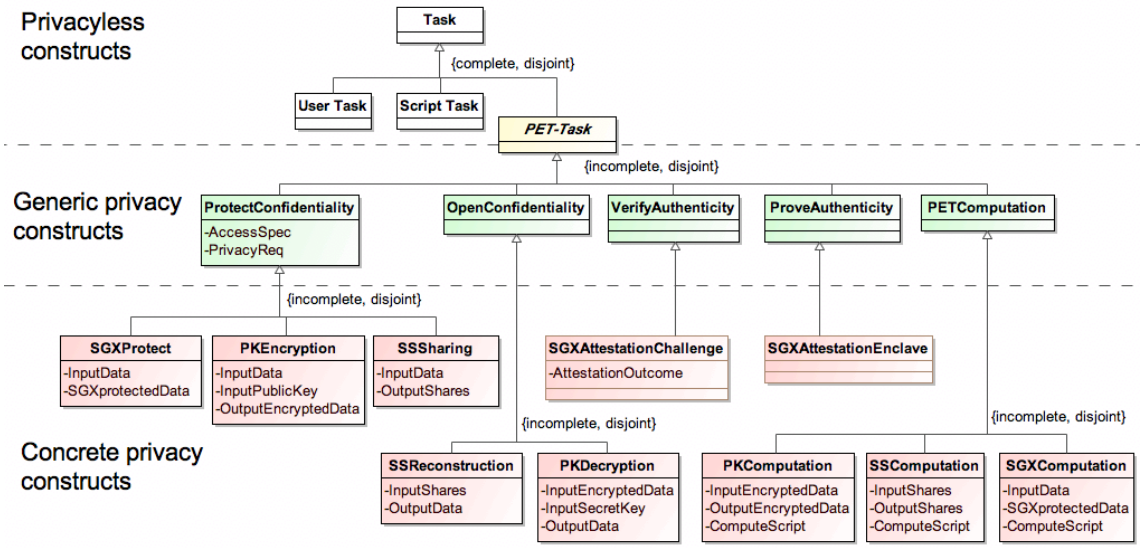


Figure 6: PE-BPMN Generic and Concrete Stereotypes

Table 3: Stereotypes for PET Goals

Group	Stereotype	Parameter
Communication	SecureChannel	Privacy requirements
Data Protection	ProtectConfidentiality	Access specification, privacy requirements
	OpenConfidentiality	
Entity Authentication	ProveAuthenticity	
	VerifyAuthenticity	
Privacy aware computation	PETComputation	Computation script

3.1.2.1 Concrete Syntax. Extension of the BPMN concrete syntax to add PETs is done using *stereotypes* with the general stereotypes given in Table 3 and their connection with some example concrete stereotypes in Figure 6. The *stereotype* characterizes the changed type of the BPMN construct. Some goals result in a series of tasks, for example data protection allows adding protection with ProtectConfidentiality and removing it with OpenConfidentiality while others have one task. The parameter describes the task, whereas the inputs and outputs should be clearly modeled as data objects or data collections in BPMN.

Access specification can, for example, specify the decryption key or the sets of shares that qualify to open the secret. Privacy requirements can specify security parameters or other properties of the scheme that should be used. The computation script is the script or query that will be executed and could be used by other analyzers in Pleak. The script fully characterizes the computations and dependency between inputs and outputs.

We consider the concrete syntax for PETs in Sec. 3.1.3 using the group stereotypes as types of the stereotype to specify which goal these stereotypes help to achieve.

3.1.3 Privacy Enhancing Technologies in PE-BPMN. This section summarizes various privacy technologies used in the Brandeis program. For each technology, we consider the participants and the tasks together with their stereotypes. For each technology we also summarize the guarantees that it gives and the assumptions that have to be satisfied for the guarantees to hold. The list of privacy stereotypes with their main details can be found in Table 4.

Table 4: Example Stereotypes

Stereotype	General stereotype	Input	Output
SecureChannel	SecureChannel	data	data
DPtask	Privacy adding	data	data
SSsharing	ProtectConfidentiality	data	shares
AddSSsharing	SSsharing	data	additive shares
FunSSsharing	SSsharing	data	function shares
PKencryption	ProtectConfidentiality	data, public key	encrypted data
SKencryption	ProtectConfidentiality	data, secret key	encrypted data
ABencryption	ProtectConfidentiality	data, public key, attributes	encrypted data
SSreconstruction	OpenConfidentiality	shares	data
AddSSreconstruction	SSreconstruction	additive shares	data
FunSSreconstruction	SSreconstruction	function shares	data
PKdecryption	OpenConfidentiality	encrypted data, secret key	data
SKdecryption	OpenConfidentiality	encrypted data, secret key	data
ABdecryption	OpenConfidentiality	encrypted data, secret key	data
SScomputation	PETComputation	shares	shares
AddSScomputation	SScomputation	additive shares	additive shares
FunSScomputation	SScomputation	function shares	additive shares
PKComputation	PETComputation	encrypted data	encrypted data
SKComputation	PETComputation	encrypted data	encrypted data
MPC	PETComputation	data	data

3.1.3.1 Secure Multiparty Computation. Secure multiparty computation (SMC) is a mean to collaboratively compute without disclosing the private inputs to other participants in the computation. It is commonly used to either outsource computations on private data or to jointly compute with private inputs of many participants. A recent overview of common secure computation techniques can be found in [8]. We have defined a stereotype MPC to denote generic secure multiparty computation, this can be specified with the concrete methods described in the following.

3.1.3.2 Secret Sharing. Secret sharing introduced by [9, 10] splits private values among participants so that some predefined groups of parties can collaboratively restore the secret. Secret sharing consists of two major tasks: producing the shares (i.e., SSsharing) from a secret and to restoring the secret from the shares (i.e., SSreconstruction). This part of secret sharing constitutes a data confidentiality mechanism. Secret sharing is most useful if it is homomorphic and allows to compute on the shares (i.e., SScomputation) allowing then privacy aware computation on confidential inputs.

Different secret sharing schemes work on different initial data types and introduce specific assumptions. A secret sharing scheme is a (t, n)-threshold scheme if the data is shared among n participants and any subset of $t \leq n$ or more participants is able to restore it. A threshold secret sharing scheme is correct if t shares uniquely determine the secret and private if any set of $t - 1$ or less shares does not give any information about the secret.

Assumptions:

- The sets of parties that can restore the secret are not colluding.

Guarantees:

- Share and any set of shares less than specified by the assumption does not leak information about the shared secret.
- Only the threshold or qualified sets of parties can restore the secret.

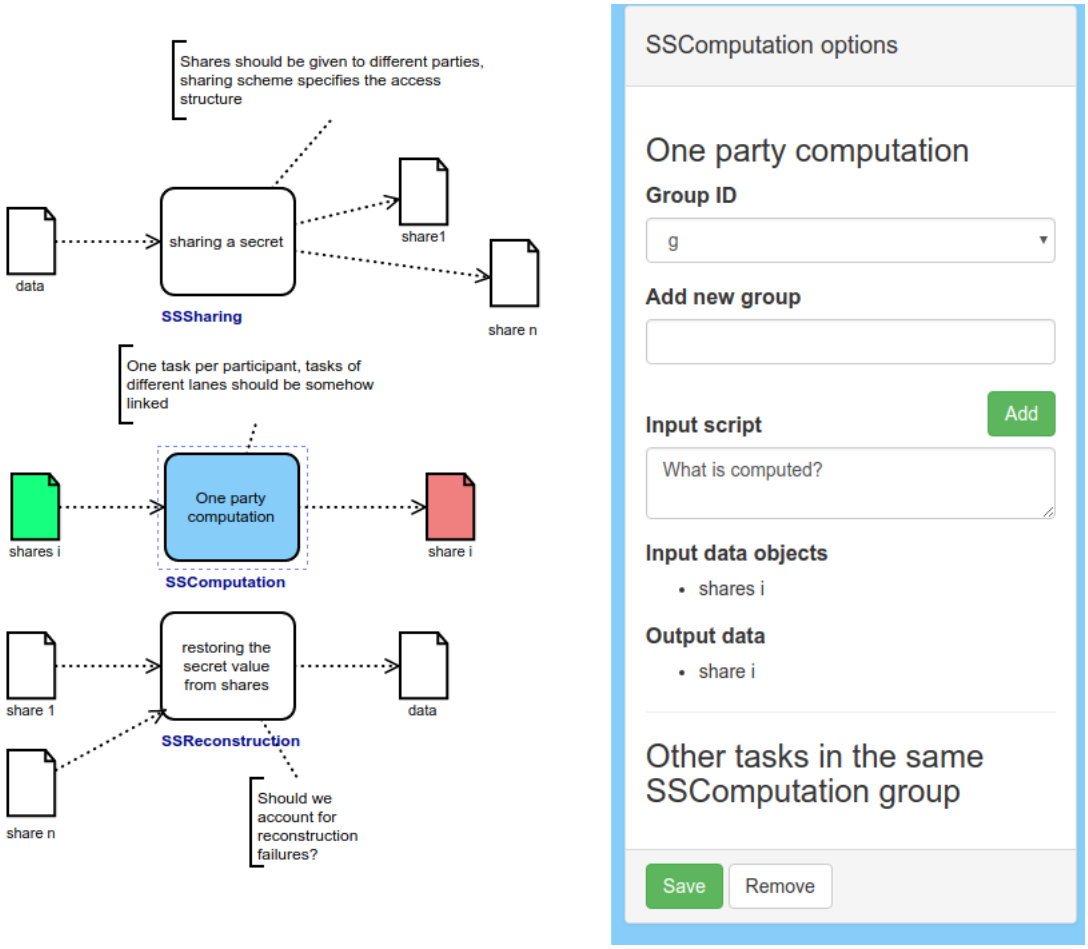


Figure 7: Secret Sharing Stereotypes and an Example of Input and Output Visualization in Pleak

- The secret can be restored from the qualified set of shares.

Figure 7 illustrates the secure computation stereotypes. SSSharing and SSReconstruction are single task stereotypes, but SSComputation is a collaborative task done by a group of participants. Hence, for this task we can form groups and give a computation script as an input. Note that all tasks in a group share the script. The figure also illustrates how the inputs and outputs of the selected task (SSComputation) are highlighted. The menu on the right is for specifying the group and script of the computations for SSComputation task.

3.1.3.3 Additive Secret Sharing.

Additive secret sharing (AddSS) is a simple (n, n) threshold secret sharing scheme on finite rings. The secrets in *AddSS* are treated as elements of a finite ring and summing all shares gives the secret. For a secret value x each participant P_i gets a randomly generated share x_i so that $x = \sum x_i$ and all parties are needed to restore the secret. The AddSSsharing generates random shares of the input x and AddSSreconstruction sums all shares of the same value together to restore the share. Moreover, AddSS is homomorphic and the parties can collaborate to compute new values from the secret shared inputs using tasks of type AddSScomputation that produces secret shared results.

AddSS inherits the assumptions and guarantees of a general perfect threshold secret sharing scheme. Note that specific schemes used for AddSScomputation may raise more specific assumptions, for example integrity protection mechanisms that have their own assumptions may be added to AddSS.

An example model for additive secret sharing can be found in `https://pleak.io/app/#/view/u2UaSnPGcAwB3yM9h6sG`.

3.1.3.4 Function Secret Sharing. *Function secret sharing* (FunSS) is a secret sharing scheme where the inputs are functions as detailed in [11, 12, 13, 14]. The general idea is that an input function is secret shared between participants and it is possible to collaboratively evaluate the function on public inputs. It is a computationally secure (n, n)-theshold secret sharing scheme.

FunSS can be thought of as an additive secret sharing scheme for functions. In FunSS the secret is a function f and the shares are also functions f_i that the participants can evaluate. Moreover, if all participants evaluate their functions f_i on a common input x then $f_i(x)$ will be the additive shares of $f(x) = \sum f_i(x)$. The function is an input to FunSSsharing that produces the function shares. In theory, it is possible to restore the function using FunSSreconstruction, but it is not commonly used as the computation results are already AddSS shares and not FunSS shares.

The computation task of stereotype FunSScomputation take as input secret shared functions and public values used to evaluate the function. The evaluation returns additive sharing. Follow-up computations can be carried out using AddSScomputation and the output can be revealed using AddSSreconstruction.

3.1.3.5 Garbled Circuits. Garbled circuits are a secure computation method for two parties. The garbler encrypts a circuit and sends it to the evaluator. The evaluator has suitable keys to decrypt the output of the circuit but can not get access to any other values in the circuit.

Essentially, the steps that either the garbler or evaluator has to carry out are asymmetric, hence we have created the respective GCGarble and GCEvaluate stereotypes. In this case the garbling task generates the circuit and the input encodings. The circuit can be sent to the evaluator however, oblivious transfer is needed to transfer only the necessary encodings. This is illustrated on Figure 8 where the GCGarble task has been selected. This task has two outputs that have distinct roles: the garbled circuit and the input encodings. The script parameter fixes the actual meaning of the circuit. However, for simplicity it is also possible to denote all this with GCcomputation stereotype assuming the computation takes in parallel for both participants.

3.1.3.6 Oblivious Transfer. Oblivious transfer is a cryptographic protocol that allows the receiver to receive one element of the sender elements so that the sender does not know which element was received. In addition, it guarantees that the receiver can not learn information about the other elements of the sender. We have created OTSEND and OTreceive stereotypes for these cases. These can be either drawn as two tasks executed in parallel or with a message flow from the sending to the receiving task.

3.1.3.7 Encryption. The modern study of encryption started with [15] and [16] lay the ground for public key cryptography. In the general case, the encryption algorithms have three tasks: key generation, encryption and decryption. However, some schemes also enable some computations on the encrypted values. We discuss the meaning of secret key and public key encryption and illustrate some cases where one can compute on the data using ciphertexts. For now we do not consider key generation and distribution.

Assumptions:

- Correct key generation.
- Correct keys are used (the key distribution is secure).

Guarantees:

- Only parties holding the correct secret key can decrypt the ciphertext.
- Encrypted text does not leak its input.
  - May leak some information (e.g length of the input).

3.1.3.8 Secret Key Encryption. Secret key encryption also known as symmetric encryption is an encryption primitive where the encryption (SKencryption) and decryption (SKdecryption) key is the

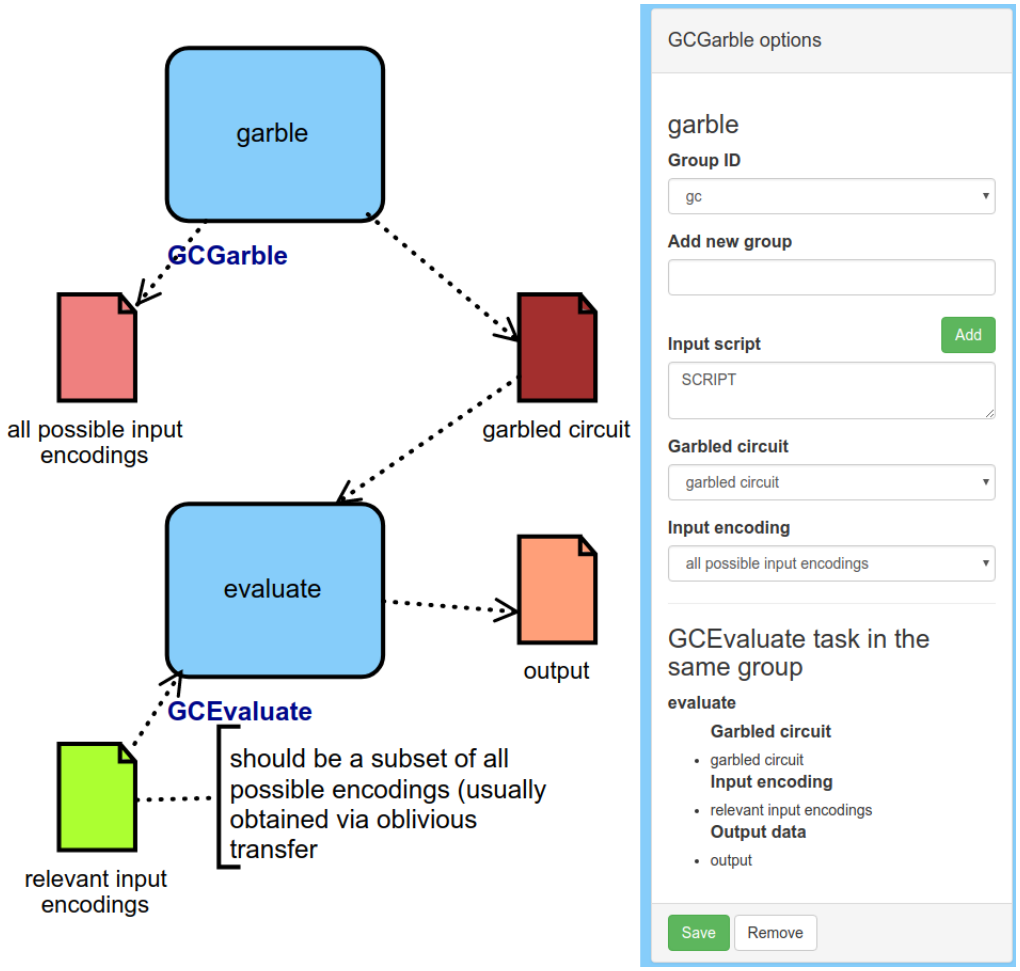


Figure 8: The Behavior of the Stereotypes in Garbled Circuit Based Secure Computation

same. For example AES is a symmetric encryption scheme. The secret key should remain private as any party knowing it also can access the protected data. Symmetric schemes are generally more efficient than asymmetric ones, but their use-cases are limited by the fact that the same key is used for both operations. All parties that can encrypt data can also decrypt all data protected with the same key. In some cases there can be some computations carried out on the encrypted data, these can be denoted by SKcomputation.

3.1.3.9 Public Key Encryption. Public key encryption schemes, a.k.a. asymmetric encryption, use a key pair or secret key and private key, where the latter can be freely distributed. Encryption (PKencryption) requires a public key and produces a ciphertext that can be decrypted (PKdecryption) with the respective secret key. Asymmetric encryption is useful in settings where many participants should encrypt some data for processing and some other set of parties (that have the secret key) can decrypt it. These two sets do not have to be the same as in the symmetric setting. In case the scheme has homomorphic properties we can also compute some operation on the ciphertexts using PKcomputaton. Public key encryption schemes are usually based on the assumption that some underlying computation (e.g. factoring integers) is difficult. Hence, considering specific schemes can introduce new assumptions.

An example model featuring these stereotypes can be found in `https://pleak.io/app/#/view/BupEXriFLsPfVxBhJ1tr`.

3.1.3.10 Homomorphic Encryption. Homomorphic encryption schemes can be both symmetric or asymmetric, with the main addition that they can perform some computation. Fully homomorphic en-

Table 5: PET Stereotypes for SGX Tasks

Stereotype	Generic stereotype	Inputs	Outputs
SGXProtect	ProtectConfidentiality	data	capsule
SGXComputation	PETComputation	capsule	capsule or data
SGXAttestationEnclave	ProveAuthentication	measurement, nonce	
SGXAttestationChallenge	VerifyAuthentication	challenge	attestation outcome
SGXQuoting	ProveAuthentication	challenge, measurement	quote
SGXQuoteVerification	VerifyAuthentication	quote, revocation list	verification outcome

cryption [17] (FHE) enables to compute all arithmetic whereas somewhat homomorphic encryption enables some more limited operations, for example, Paillier cryptosystem [18] allows addition of encrypted data. To stress the used computation scheme it is possible to use specific stereotypes, such as FHEcomputation for computation on FHE scheme. Note that such computation expects the encrypted inputs to be generated with a scheme that is a FHE scheme. The more general stereotypes SKcomputation and PKcomputation can be used if the homomorphic scheme is not yet specified, on the other hand we can also specify the encryption (FHEencryption) and decryption (FHEdecryption) to make the models more explicit.

3.1.3.11 Attribute Based Encryption. Attribute based encryption is like public key encryption with several private keys and attributes added. The public key defines all possible attributes and the secret keys contain the attributes of the user. In every encryption operation the attributes that are allowed to decrypt the resulting ciphertext are fixed.

The stereotypes for attribute based encryption reflect those of the public key encryption but allow for multiple different secret keys and fixing attributes to the keys. Respectively we define stereotypes ABencrypt and ABdecrypt for encryption and decryption tasks respectively. In addition for the keys we define data stereotypes ABpublic and ABprivate. Each private key is joined with one public key where public key defines all possible attributes and a private key fixes a subset of these attributes.

3.1.3.12 Software Guard Extension. Intel Software Guard Extension (SGX) introduced in in [19] and [20] is a technology for secure computation with the help of special processor instructions. The code is loaded into a secure enclave and executed on encrypted data. Our approach on SGX stereotypes is summarized in Table 5.

Assumptions:

- Intel SGX Attestation Service is trusted.
- The computations are side-channel safe (otherwise the side-channels specify some leakage).
- SGX machines are attested.

Guarantees:

- All computations are carried out according to the code.
- All data is processed according to the script.
- Only the explicitly published outputs are leaked, all intermediate values are confidential.
- The attestation process guarantees that the attested machine is a SGX machine and establishes a key for a secure channel with the enclave.

SGX Computation. SGX computation is a version of secure computation with the guarantees enforced by special purpose hardware instructions. The general idea is that the private data is processed inside an enclave where it can be decrypted for processing, but the processing is protected by memory-encryption.

The inputs to the computation can be provided as both public values and encrypted with the public key of the enclave. For private inputs, we use the SGXProtect stereotype to denote encryption with the assumption that SGXcomputation stereotype can use the respective decryption key and encrypted inputs to produce outputs without leaking the encrypted inputs to the party running the enclave.

The assumptions required for the security of SGX computation are as follows:

- The computation runs on SGX processor.
- The computation runs the expected algorithm.
- The encryption key is generated correctly.

All of these can be verified using the attestation process, hence adding attestation to the models can leverage some assumptions that result from the use of SGXcomputation.

An example of the SGX computation combined with public key encryption can be seen in `https://pleak.io/app/#/view/IYrBIt0hcGdZgDZGpUNV`.

SGX Attestation. The attestation process is a cryptographic protocol outlined in [21] and [22] that gives various guarantees. Firstly, it enables to check that the enclave is running the same software as was initially loaded to the enclave. Secondly, it enables to verify that the code is indeed running in an enclave of a processor with SGX capabilities. Finally, the process also establishes a secure communication channel between the attestation challenger and the enclave. The general process is the same independently of the scenario and many different stakeholders of business processes can be in the role or the attestation challenger or the enclave. We can use the SGXattestation stereotype on the challenger side and the SGXquoting stereotype of the side of the enclave to abstract away the full process. These two stereotypes require an introduction of a external Intel SGX attestation service stakeholder that has the SGXquoteVerification stereotype.

The attestation has these main assumptions:

- Intel SGX processor design is secure.
- Intel attestation service is trusted and the revocation list is maintained properly.

It is possible to model the attestation process itself using the required PETs for reporting and signing to go more into the details of the assumptions of the attestation.

The attestation is a version of authentication where the attestation challenger who executes SGXattestation type task gains guarantees about the enclave.

3.1.3.13 Protected Communication. Communication protection combines many tools like encryption and authentication mechanisms. It can also accommodate anonymous communications, however this is likely to introduce a larger context of other parties in the network. Hence, this category covers onion routing and mixing networks as well as TLS or IPSec technologies. Network communication mostly passes through many external parties that route the traffic, hence we consider all message flows to pass through an external telecommunications provider that sees the network communication. At the moment we only consider the two classes (secure and anonymous) and do not focus on the actual protection technologies.

Secure Channel. We introduce one stereotype SecureChannel that specifies the message flow to be secure. The understanding should be that the sender secures the message, it is transmitted through the network without the possibility to either see the content or modify the integrity and the receiver is able to remove the protection and use the sent element. In addition, many concrete technologies begin with a protocol to agree on the communication parameters. However, we feel that in most processes the possibility to denote flows as secure and to specify the parameters in which respect the security is ensures is enough. It would be possible to remove this stereotype and introduce the respective joint stereotypes for the sender, receiver and the telecommunication party.

Guarantees:

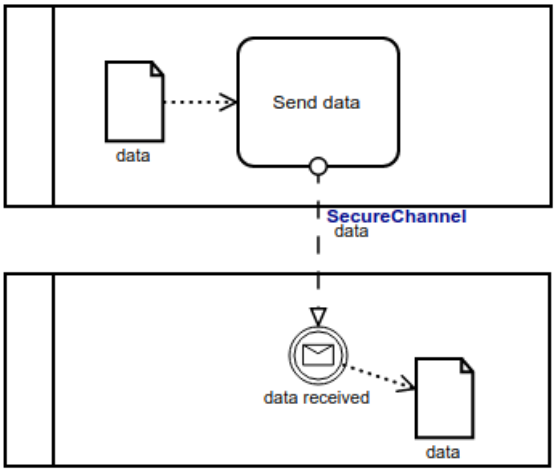


Figure 9: Usage of SecureChannel Stereotype

- The transferred data is only revealed to the intended recipient.
- The data can not be modified while in transit.

The SecureChannel stereotype is illustrated on Figure 9. The stereotype itself appears next to the data flow that denotes the network communication. The intended usage is that the input to the sending task and the output of the receiving event are the same data object. The name of the transferred data object can also be reflected on the name of the data flow to make it clearer. One participant holds the data and performs a task of sending it, the receiving participant receives the same data as was sent. No eavesdropper on the network learns the data.

3.1.4 PE-BPMN Implementation. Not all stereotypes can be added to all tasks, for example there has to be suitable number of inputs and outputs. For some stereotypes, it can be that there are special roles that the inputs or outputs have. For example, an encryption operation has two distinct inputs - the key and the plaintext - that can be identified on the model. For some stereotypes the only restriction is that there must be a certain number of inputs or outputs. Table 6 summarizes the implemented restrictions for currently used stereotypes. It lists the number of expected inputs and outputs as well as parameters. In case the inputs or outputs have special roles, then they are also named in the table and the user interface allows to fix which data object has the specified role.

Table 6: Restrictions and Parameters of the Implemented Task Stereotypes

Stereotype	Inputs	Outputs	Other parameters
ProtectConfidentiality	1: data	1: protected data	
SGXProtect	1: data	1: enclave data	group with SGX-Computation
PKEncrypt	2: public key, data	1: ciphertext	
SKEncrypt	2: secret key, data	1: ciphertext	
ABencrypt	2: public key, data	1: ciphertext	attributes of the keys that can decrypt
SSSharing	1: data	2 – . . .: shares	number of shares, threshold
AddSSSharing	1: data	2 – . . .: additive shares	
FunSSSharing	1: function	2: function shares	

OpenConfidentiality	1: protected data	1: data	
PKDecrypt	2: private key, ciphertext	1: data	
SKDecrypt	2: secret key, ciphertext	1: data	
ABdecrypt	2: secret key, ciphertext	1: data	
SSReconstruction	2 – . . .: shares	1: secret data	
AddSSReconstruction	2 – . . .: additive shares	1: secret data	
FunSSReconstruction	2: function shares	1: secret function	
PETComputation	1 – . . .: protected data, data	1: protected data or data	script
SGXComputation	1 – . . . : enclave data, data	1 – . . . : enclave data or data	script, group with SGXComputation, group with SGXProtect, group with SGXAttestationEnclave
PKComputation	1 – . . . : ciphertexts, data	1 : ciphertext	script
SKComputation	1 – . . . : ciphertexts, data	1 : ciphertext	script
MPC	0 – . . . : data (at least one task in group has an input)	0 – 1: data (at least one task in group has an output)	script, grouped with MPC
SSComputation	1 – . . . : shares, data	1: share	script, grouped with SSComputation
FunSSComputation	2: function share, evaluation point	1: additive share	grouped with FunSSComputation
AddSSComputation	1 – . . . : additive shares, data	1: additive share	script, grouped with AddSSComputation
GCGarble	0:	2: garbled circuit, input encodings	script, grouped with GCEvaluate
GCEvaluate	2: garbled circuit, input encodings	1: computation output	grouped with GCGarble
OTSend	1: input data	0:	grouped with OTReceive
OTReceive	1: query	1: input	grouped with OTSend
SGXAttestationEnclave	0-...: enclave measurement	0:	group with SGXAttestationChallenge, group with SGXComputation
SGXAttestationChallenge	0-...: challenge	1: attestation outcome	group with SGXAttestationEnclave
SGXQuoting	2: challenge, measurement	1: quote	
SGXQuoteVerification			

3.1.4.1 Type Checking. The stereotypes have specific integrity constraints that should be followed for the privacy model to be syntactically correct. They require inputs and generate outputs that need to be consistent to capture the meaning of the stereotyped activity. For instance, PKEncrypt requires an input

of data in plaintext and a publicKey and results in a ciphertext, encryptedData. We expect that the public key input to PKEncrypt is fixed as PKPublic. In addition, for many stereotypes we also need to verify that the input is indeed of the type claimed on the model. e.g. that an input to PKDecrypt has indeed come from PKEncrypt or PKComputation and is a ciphertext. Table 6 lists the input and output types that the tasks expect and it can be used as a reference to see which sequences of stereotypes are valid on the model. The label *data* can apply to any data object on the model meaning that there are no restrictions on the inputs and the output is treated as having no protection mechanism. However, the table does not list all restrictions that need to hold. For example, for PKDecrypt we also require that the private key (of type PKPrivate) that is used to decrypt forms a key pair with the public key (of type PKPublic) that was used to initially encrypt the data. For computations of PKComputation type we expect that all ciphertext inputs correspond to encryptions using the same public key and the output is then also considered to use the same key.

In short, it can be said that the protection mechanism (or protection type task) on the model limits the correct processing of the protected data and can define parameters that need to be checked in computation or opening phases. Table 7 lists the conditions that need to hold for the opening stereotypes to make protected data public or for the computation stereotypes to be able to perform the computations.

3.1.4.2 Grouped Stereotypes. Stereotypes that belong to groups usually have restrictions to which tasks there need to be in a group. Groups denote computations that somehow belong together. Mostly, we group stereotypes that correspond to separate tasks of collaborative protocols. For example, MPC tasks are grouped together.

For Intel SGX technology where we group all tasks carried out in a single enclave. For example, tasks with SGXAttestationChallenge stereotype must come in pairs with a task with SGXAttestationEnclave stereotype, while this SGXAttestationEnclave stereotype task can also be in a group with multiple SGXComputation tasks. We expect the SGXAttestationEnclave and SGXComputation to be carried out by the same enclave if grouped together.

Table 8 summarizes the restrictions that apply to groups of different stereotypes. For tasks that need to be executed in parallel we also require that they are executed by different stakeholders.

We are also covering minor BPMN standard checks for some stereotypes. For example, we confirm the presence of a start event. If negative, we are unable to check parallelism (concerning reachability of tasks) in models with gateways when we have no start event to begin the check from.

Validation results are reported as a list of errors (colored red) and warnings (colored orange) or as a success message "Passed validation". While warnings are permitted, it is required that there are no errors in the model to run the analysis outlined in the following.

3.1.5 PE-BPMN Privacy Analysis.

3.1.5.1 Simple Disclosure. Disclosure analysis summarizes who sees which data object and whether or not they have access to the contents of the data. Simple disclosure report is a table where columns are data objects from the process and rows are the stakeholders (lanes). Each cell is marked either V (visible), H (hidden) or –. Marking – means that this stakeholder does not see this data object in the process. On the other hand V means that the contents of this data are fully visible to the stakeholder. H is the middle ground denoting that the participant has the data object, but it has a form of protection on it (e.g. some type of ProtectConfidentiality or PETComputation taks produced it). For example, a ciphertext will be denoted with H. This analysis is performed by traversing the model graph and tracking the states of the data objects.

3.1.5.2 Data Dependency. Simple data dependency gives the data dependency matrix of the model. The relations described there are either straightforward from the model data associations or result from collaborative (grouped) tasks. Essentially the data dependency analysis gives an adjacency matrix for the process from the viewpoint on the data in the process. We mark D (direct dependency) for cases

Table 7: Conditions for Allowed Opening and Computation Stereotypes

Stereotype	Success conditions
OpenConfidentiality	input comes from ProtectConfidentiality or PETComputation
PKDecrypt	ciphertext input comes from PKEncrypt or PKComputation, uses the PKPrivate corresponding to the PKPublic used by the input
SKDecrypt	ciphertext input comes from SKEncrypt or SKComputation, uses the same secret key as the input
ABDecrypt	ciphertext input comes from ABencrypt, uses a ABpublic corresponding to the ABprivate key used for decryption and the decryption key has at least one of the attributes fixed in the ABencrypt task
SSReconstruction	all input shares correspond to the same secret shared value (outputs of one SSSharing or one group of SSComputation), at least *threshold* shares available
AddSSReconstruction	all input shares correspond to the same secret shared value (outputs of one AddSSSharing or one group of AddSSComputation or FunSS-Computation), all shares available
FunSSReconstruction	all input shares correspond to the same secret shared value (outputs of one FunSSSharing), both shares available
SGXComputation	enclave data comes from SGXProtect or as enclave data from SGX-Computation in the same group
PKComputation	ciphertext inputs come from PKEncrypt or PKComputation, all ciphertext inputs correspond to the same PKPublic public key, the output corresponds to the same key
SKComputation	ciphertext inputs come from SKEncrypt or SKComputation, all ciphertext inputs correspond to the same secret key, the output corresponds to the same key
SSComputation	share inputs come from SSSharing or SSComputation tasks, all shared inputs have the same threshold, the output has the same threshold as the shared inputs
FunSSComputation	function share input from FunSSSharing, output is two party additive secret shared value
AddSSComputation	additive share inputs from AddSSSharing or AddSSComputation tasks, all additive share inputs have the same number of shares, output is additive secret sharing for the same number of shares as the inputs

where data A is an input to a task that produces data B - meaning that B directly (through one task) depends on A. If data C in turn depends on B then we mark I (indirect dependency) for the dependency between A and C - C indirectly (through a path of more than one task) depends on A. This analysis is performed by traversing the model graph.

3.1.5.3 Extended Disclosure. Extended disclosure combines simple disclosure and data dependency. In addition to the visibility this allows to get a glimpse of the consequences of some data becoming visible for some party. Essentially, for any marker V in the simple disclosure we look at the data dependency to see which data this object depends on. Making this data visible to some party has a risk of leaking something about the data that it depends on. Other layers of analysis, e.g. leaks-when and sensitivity analysis can then be used to study this risk in more detail.

For any data object in the model each participant may have a subset of V, H, -, I and D annotations. If there is V then it is clear that the participant has full access to the given data. Analogously when there is only - or H then the participant does not have access to said data. However, a combination of -/H with

Table 8: Restrictions for Stereotype Groups

Stereotype	# tasks	Restrictions
MPC	2 – ...	Parallel tasks, at least one input per group, at least one output per group
SSComputation	2 – ...	Parallel tasks, at least one shared input, all tasks have the same number of inputs and outputs, the number of tasks is the number of shares in each shared input values, the same data inputs for all tasks, each task has one distinct share input for each shared input
FunSSComputation	2	Parallel tasks, one shared input, the same evaluation point input for both tasks
AddSSComputation	2 – ...	Parallel tasks, at least one shared input, all tasks have the same number of inputs and outputs, the number of tasks is the number of shares in each shared input values, the same data inputs for all tasks, each task has one distinct share input for each shared input
SGXComputation, SGXProtect, SGXAttestationEnclave	1 – ...	All group tasks are on the same lane (same CPU executes them)
SGXAttestationChallenge, SGXAttestationEnclave	2	Parallel tasks, both exist
OTSend, OTReceive	2	Parallel tasks, both exist, may have a message flow from OTSend to OTReceive
GCComputation	2	Parallel tasks, at least one input per group, at least one output per group
GCGarble, GCEvaluate	2	Both exist, on separate lanes, GCGarble garbled circuit output is the garbled circuit input to GCEvaluate, GCGarble input encodings output is used to derive encoded input for GCEvaluate

D and I means that while the participant does not have direct access to the data it does see something that is derived from this data object. Hence, there is a possibility that something about this data leaks to the said participant. These are the cases that should be studied further with leaks-when or sensitivity analysis to discover which information about the data is actually leaked to the party.

3.1.5.4 Leakage Detection. Leakage detection [23] is used to analyze more complex PE-BPMN models where the disclosure tables are not sufficient to get a good overview of the process (e.g. the models with a lot of branching). It is used to detect if some input data may end up in certain points of the model - e.g. some task or participant. Leakage detection takes into account the possible executions of the process over the different branching choices (but satisfying the synchronization rules of grouped tasks). For example, this allows us to find if some party has many shares of secret shared value but no explicit task to reconstruct this value. Analogously, we can discover if there are possible runs of the process where some party is not explicitly required to have a ciphertext and a respective decryption key has them.

Each party is translated into a transition system, encoding which tasks are executed in which order, and where are the choices governing the selection of the tasks to execute. The states of this transition system are then extended with the data objects known to each party at the execution point corresponding to this state. The transitions are extended with the data flows inside and between parties. Concretely,

PE-BPMN models are transformed to a specific type of process algebraic specification language, namely the *micro Common Representation Language*[8] (*mCRL* for short). *mCRL* is a specification language that extends the Algebra of Communicating Processes (ACP) [24] by adding important feature for modelling of real systems like data, time and multi-actions [25, 26]. This language is also equipped with a toolset that provides equivalence and model checking over the specification.

Our queries ask whether states with certain properties are reachable. The technical details of the leakage analyzer can be found in [23]. Given the statement of a property expressing that certain data objects together reveal a secret to a party not entitled to this secret (e.g. no explicit recovery of this secret is on this model), we use the tools from the mCRL2 toolset to model-check this specification. Model checking is supported by this property description (stated in the μ-calculus) for each of the properties. For different PETs supported in PE-BPMN, the properties we analyse capture different types their misuses. We have found *mCRL* particularly well-suited for the purposes of model checking of BPMN and PE-BPMN models, because it naturally captures the notion of "independent processes" that communicate via message exchanges, which is the core of a PE-BPMN collaboration diagram.

The main goal of leakage detection is to indeed discover unintended leakages, especially since in complex processes these could occur easily. However, in case any leakage is detected the user has to decide if it is a real leakage or an error in the model. For example, a possible leakage to a party who should indeed get the data could indicate either a missing OpenConfidentiality type stereotype or some inefficiency on the model if the protection is removed later than possible.

3.2 Leaks-When Analysis

We call our qualitative analyses "Leaks-when analysis", because they explain, what leaks and when. Here "what" refers to the components of data objects that are inputs to the processes. "Leaks" refers to the outputs of the process having dependency on the inputs. "When" means, that we characterize, under which conditions this dependency manifests. To be applicable to leaks-when analyses, the tasks of the business processes must have sufficiently precise descriptions of what operations they apply on their inputs in order to produce their outputs.

3.2.1 SQL Workflows. In the context of the NAPLES project, a *SQL workflow* is a sequence of SQL statements, intended to be executed against a database with a certain schema. Each SQL statement in the workflow produces a new dataset with a certain schema, which is given a name, and which the subsequent SQL statements can use as tables which have been added to the database. The dataset produced by the last SQL statement in the workflow is the result of the entire workflow. The goal of the leaks-when analysis is to characterize, how the contents of the database affects the result of the workflow — which attributes of which tables become part of the result, in which manner, and under which conditions.

Our analysis is based on precisely connecting the definitions and uses of various quantities, starting from the attributes of the tables of the input database, going through the computing operations of each of the SQL-statements, and ending in the attributes of the resulting dataset. We try to overestimate the dependencies as little as possible. The internal representation that we use for the dependencies must support the detection of simplification possibilities, such that spurious dependencies can be removed.

As the execution of SQL workflows is mostly data-oriented (as opposed to control-oriented, which would be the case for more imperative-style specifications), we are looking for a graph-based representation, where each node corresponds to an input, to an output, or a simple computation. The arcs connect the nodes producing a value with the nodes that use this value. The graph has to have a well-defined semantics matching with the semantics of SQL workflows, giving the justification for a simplification being allowed. Most of the simplification possibilities would be detected by looking at small subgraphs of the whole graph, and replacing them with another small subgraph.

[8] www.mcrl2.org

$$
\begin{array}{rcl}
Q & ::= & t \\
& | & Q_1 \times \cdots \times Q_k \\
& | & [Q]_{a \to a'} \\
& | & \sigma(Q; e) \\
& | & \pi_{a_1,\ldots,a_k}(Q) \\
& | & \mathsf{col}_{a \leftarrow e}(Q) \\
& | & \mathsf{let}\ t = Q_1\ \mathsf{in}\ Q_2
\end{array}
\qquad
\begin{array}{rcl}
e & ::= & a \\
& | & \otimes(e_1, \ldots, e_k)
\end{array}
$$

Figure 10: Syntax of Queries

We have used such data dependency graphs before, to perform computationally sound analysis of cryptographic key-exchange protocols [27, 28]. Our experience with this representation for cryptographic protocols has been mixed. The focus on data dependencies was definitely helpful in detecting where a game-based security definition (stating that two rather simple computations look the same to an adversary) of a cryptographic primitive may be applied. On the other hand, the description of the "obviously secure" computation introduced control dependencies that we could not fully satisfactorily handle. Nevertheless, for certain symmetric-key protocols we could show message secrecy — after applying the security definition of symmetric encryption, and simplifying the resulting graph, the input message was no longer used to construct protocol messages that were seen by the adversary.

In SQL workflows, we expect to have even less control dependencies, and finer data dependencies. Hence we have extended our dependency graphs to argue about the computations in these workflows.

3.2.1.1 Databases, Schemas, and Queries. A *relation schema* is $r(a_1 : D_1, \ldots, a_n : D_n; \mathsf{Dis}_r)$, where r is relation name, $a_1, \ldots, a_n$ are attribute names, $D_1, \ldots, D_n$ are sets, and Dis_r is a set of subsets of the set of attributes $\{a_1, \ldots, a_n\}$. The last component indicates, which attributes or sets of them must be unique in a relation satisfying this schema. An element of Dis_r describes a possible index for a table satisfying the relation schema r. In our analysis, we require Dis_r to contain at least one set of attributes.

Let $D[r]$ denote the set $D_1 \times \cdots \times D_n$. A relation R over the schema r is a subset of $D[r]$, such that for each $\{a_{i_1}, \ldots, a_{i_k}\} \in \mathsf{Dis}_r$ and each $(x_{i_1}, \ldots, x_{i_k}) \in D_{i_1} \times \cdots \times D_{i_k}$ there is at most one $(y_1, \ldots, y_n) \in R$ satisfying $y_{i_1} = x_{i_1}, \ldots, y_{i_k} = x_{i_k}$. Let $\mathcal{X}_r$ denote the set of all relations over the schema r. For $x \in D[r]$, let $x[a_i]$ denote the value of attribute a_i on x.

A *database schema* is $dbs = (t_1 : r_1, \ldots, t_m : r_m)$, where $t_1, \ldots, t_m$ are *table names* and $r_1, \ldots, r_m$ are relation schemas. A database over the schema is a tuple of relations $\mathbf{D} = (R_1, \ldots, R_m)$, where R_i is over r_i. For a fixed dbs, let $\mathcal{Y}$ denote the set of all databases over the schema dbs, and let $D[t_i]$ denote the set $D[r_i]$. For a database $Y \in \mathcal{Y}$, let $Y.t_i \subseteq D[r]$ denote its table t_i.

Suppose that we have selected the primary keys for each table in the database. That means, for each $t : r$ in the database schema, we have selected $\mathsf{index}_r \in \mathsf{Dis}_r$. We can then think of a relation R over the schema $r(a_1 : D_1, \ldots, a_n : D_n; \mathsf{Dis}_r)$ as a set of partial functions $\mathsf{f}^r_1, \ldots, \mathsf{f}^r_n$ from the cartesian product $\prod_{a_i \in \mathsf{index}_r} D_i$ to each of the sets $D_1, \ldots, D_n$. All these partial functions are defined on the same domain. If $a_i \in \mathsf{index}_r$, then the function f^r_i must be a partial projection.

The syntax for workflows of simple database queries is given in Fig. 10. The workflow is executed against a database with a certain schema dbs. The meaning of the syntax for queries Q is the following.

- The query t returns the table t. This table must exist in the current database.
- The query $Q_1 \times \cdots \times Q_k$ returns the Cartesian product of the results of the queries $Q_1, \ldots, Q_k$, which are executed against the current database. We require that the names of the attributes in $Q_1 \times \cdots \times Q_k$ are unique, i.e. the queries $Q_1, \ldots, Q_k$ result in datasets which have non-intersecting sets of attributes.
- $[Q]_{a \to a'}$ executes the query Q. Its result is a relation with a certain schema; this schema must contain the attribute a. This attribute is then renamed to a'.

- $\sigma(Q; e)$ filters the result of the query Q with the expression e. The expression e, which must return a Boolean value, is built up from attributes and arithmetic / relational / logical etc. operations $\otimes$. We expect the expressions e to be well-typed, but will not discuss this here any more.
- $\pi_{a_1,\ldots,a_k}(Q)$ projects the result of Q onto attributes $a_1, \ldots, a_k$. The dataset returned by Q must have these attributes in its schema.
- $\mathsf{col}_{a \leftarrow e}(Q)$ runs Q and then adds a new column (a new attribute) to the result. The name of the attribute is a. Its value for each row is computed from the existing attributes of this row according to the expression e.
- $\mathsf{let}\ t = Q_1\ \mathsf{in}\ Q_2$ is used to build workflows. It executes the query Q_1 against the current database and gives the resulting dataset the name t. It will then execute the query Q_2 against the database the contains the current database, as well as the the table t.

We see that the available operations correspond to the steps in SELECT ... FROM ... WHERE ... queries. In the following sections, we will consider more operations, eventually arriving at a set where we can express the *aid distribution* workflow from the Enterprise CRT.

3.2.1.2 Dependency Graphs. A *dependency graph* (DG) is a directed graph $G = (V, E, \mathsf{s}, \mathsf{t}, \ldots)$, where $\mathsf{s}, \mathsf{t} : E \to V$ give the source and the target nodes of arcs. The DG also has the following additional components:

- There are subsets of nodes $I, O \subseteq V$. The in-degree of any node in I and the out-degree of any node in O is 0. The in-degree of any node in O is 1. These nodes represent the inputs coming to, and the outputs produced by the DG.
- There is a set **Op** of possible operations. Each *internal* node v (i.e. $v \in V \backslash (I \cup O)$) has a label $\lambda(v) \in \mathbf{Op}$.
- For each internal node v, its incoming arcs are linearly ordered; let $<_v$ denote the ordering relation. The number of incoming arcs of an internal node v is equal to the number of operands that the operation $\lambda(v)$ expects.

Let $\mathbf{V}$ be a set of values; the operations in **Op** consume and produce values. Given the semantics $[\![\otimes]\!] : \mathbf{V}^* \to \mathbf{V}$ of each operation $\otimes \in \mathbf{Op}$, the dependency graph G defines a mapping $[\![G]\!] : \mathbf{V}^I \to \mathbf{V}^O$. If G has no directed cycles, then this mapping is defined by assigning a value to each node of G, with the values for input nodes given by the input to $[\![G]\!]$; the values of intermediate nodes v computed by applying $\lambda(v)$ to the values of direct ancestors of v; and the values of output nodes being equal to the values of their direct ancestors. For dependency graphs with directed cycles, the semantics can be defined using a fix-point construction [27], if there is a partial order on $\mathbf{V}$ with the least element $\perp$, and if the operations are monotonic. As we do not have cyclic dependency graphs here, we will not discuss this any more.

A dependency graph may be infinite, with infinitely many inputs and outputs, as well as with nodes having an infinite number of incoming edges. In the latter case, the operation in the node must make sense for infinite number of inputs (e.g. it may be conjunction or disjunction of booleans). If G is infinite then $[\![G]\!]$ is still well-defined as long as for each output node v_O there is a bound B_O, such that any path in the graph ending in v_O has length at most B_O.

The computations of an SQL workflow can naturally be expressed as infinite dependency graphs. Given a table t with the schema $r(a_1 : D_1, \ldots, a_n : D_n)$ and its index index_r, we express its use in a workflow by the input nodes $v^t_{i,K}$ for each attribute a_i and each possible value K of the index attributes of t. Additionally, the use of the table t is expressed by the input nodes $v^t_{\exists,K}$, denoting whether the row with the index value K is present in the database. As the index attributes typically come from infinite sets (e.g. integers), there are infinitely many possible values K. The input nodes $v^t_{i,K}$ and $v^t_{\exists,K}$ are followed by computation nodes for the expressions e occurring in the workflow. Again, these are replicated as many times as there are possible values for index attributes in the relations that they work on. We end up with a graph with output nodes $w_{j,K'}$ and $w_{\exists,K'}$ for each possible value K' of the index of the resulting

Q	index_Q
t	$\prod_{a_i \in \mathsf{index}_r} D_i$, where $r(a_1 : D_1, \ldots, a_n : D_n)$ is the schema of t
$Q_1 \times \cdots \times Q_k$	$\mathsf{index}_{Q_1} \times \cdots \times \mathsf{index}_{Q_k}$
$[Q]_{a \to a'}$	index_Q
$\sigma(Q; e)$	index_Q
$\pi_{a_1,\ldots,a_k}(Q)$	index_Q
$\mathsf{col}_{a \leftarrow e}(Q)$	index_Q
$\mathsf{let}\ t = Q_1\ \mathsf{in}\ Q_2$	index_{Q_2}, where $\mathsf{index}_t \leftarrow \mathsf{index}_{Q_1}$

Figure 11: Computing the Index Set of the Query

dataset. The attributes of the index of the resulting dataset, and hence also the set from which the values K' come from, can be computed from the query as shown in Fig. 11.

We represent the infinite dependency graphs as finite summaries. The summary dependency graph (SDG) has the same components $(V, E, I, O, \lambda, <)$ as a DG. However, there is additional structure for the nodes and the edges, as described below.

- There is a set of possible index sets $\mathcal{S}$. The elements of $\mathcal{S}$ are typically the set of integers, the set of strings, the unit set (a set with a single element). For handling a particular database schema, $\mathcal{S}$ must contain all sets D_i that are associated to some attribute in the index of some table in this schema.
- Each node $v \in V$ has the *dimension* $\mathsf{dim}(v)$ and *input dimension* $\overrightarrow{\mathsf{dim}}(v)$. They are both sets.
  - In our representation, both $\mathsf{dim}(v)$ and $\overrightarrow{\mathsf{dim}}(v)$ are sets that can be expressed as *polynomials* over $\mathcal{S}$. A polynomial over a set of sets $\mathcal{X}$ is a set of the form $\sum_{i=1}^{n} \prod_{j=1}^{m_i} X_{ij}$, where $X_{ij} \in \mathcal{X}$, and $\sum$ denotes the non-intersecting union (or: sum) of sets. Hence there is a finite representation for $\mathsf{dim}(v)$ and $\overrightarrow{\mathsf{dim}}(v)$.
- Each node v has a mapping $\delta(v)$ from $\overrightarrow{\mathsf{dim}}(v)$ to $\mathsf{dim}(v)$.
  - In our representation, the mapping $\delta(v)$ is a *canonical polynomial map*. Let $\overrightarrow{\mathsf{dim}}(v) = \sum_{i=1}^{n} \prod_{j=1}^{m_i} X_{ij}$ and $\mathsf{dim}(v) = \sum_{i=1}^{s} \prod_{j=1}^{t_i} Y_{ij}$. A canonical polynomial map is built up from identity mappings between X_{ij} and $Y_{i'j'}$ (which must be the same set) as follows:
    - A canonical mapping $c : \prod_{j=1}^{m} X_j \to \prod_{j=1}^{t} Y_j$ is defined by an injective mapping $\gamma : \{1, \ldots, t\} \to \{1, \ldots, m\}$ satisfying $X_{\gamma(j)} = Y_j$ for all $j \in \{1, \ldots, t\}$. The mapping c is given by
    $$c((x_1, \ldots, x_m)) = (x_{\gamma^{-1}(1)}, \ldots, x_{\gamma^{-1}(t)}) \ .$$
    - A canonical mapping from $\prod_{j=1}^{m} X_j$ to $\sum_{i=1}^{s} \prod_{j=1}^{t_i} Y_{ij}$ consists of an index $q \in \{1, \ldots, s\}$ and a canonical mapping of the previous kind from $\prod_{j=1}^{m} X_j$ to $\prod_{j=1}^{t_q} Y_{qj}$.
    - A canonical mapping from $\sum_{i=1}^{n} \prod_{j=1}^{m_i} X_{ij}$ to $\sum_{i=1}^{s} \prod_{j=1}^{t_i} Y_{ij}$ consists of n canonical mappings of the previous kind.
  - If $\delta(v)$ is not the identity mapping, then the node v must have exactly one incoming arc.
- Each arc $\alpha \in E$ still has a single target node $\mathsf{t}(\alpha)$. But an arc may have several source nodes, i.e. $\mathsf{s}(\alpha) \subseteq V$.
- Each arc $\alpha \in E$ has a mapping $\overline{\delta}(\alpha)$ from $\overrightarrow{\mathsf{dim}}(\mathsf{t}(\alpha))$ to $\sum_{v \in \mathsf{s}(\alpha)} \mathsf{dim}(v)$.
  - In our representation, the mapping $\overline{\delta}(\alpha)$ is again a canonical polynomial map.

A summary dependency graph G_{sum} is expanded to a potentially infinite dependency graph $G = \mathsf{expand}(G_{\mathrm{sum}})$ in the following manner:

- For each node v in the summary dependency graph, there are nodes $\{(v, x) \mid x \in \mathsf{dim}(v)\}$ in the actual dependency graph. They all have the same operation $\lambda(v)$.

– We call the node (v, x) in the actual dependency graph the *instance* x of the node v in the SDG.

- For each arc α going to a vertex v in the summary dependency graph, and for each element $x \in \overrightarrow{\mathsf{dim}}(v)$, there is an edge from the node $\overline{\delta}(\alpha)(x)$ to the node $\delta(v)(x)$. Note that the output of $\overline{\delta}(\alpha)(x)$ is a pair of some node $w \in \mathsf{s}(\alpha)$ and a value $y \in \mathsf{dim}(w)$.
  - Let $x \in \mathsf{dim}(v)$. If $\delta(v)$ is the identity mapping and thus $v \in G_{\text{sum}}$ may have several input arcs, the ordering $<_{(v,x)}$ of the inputs of the vertex $(v, x) \in G$ is inherited from v. The vertex (v, x) has the same number of input arcs as the vertex v does.
  - Otherwise, the vertices $(v, x) \in G$ may have any number of inputs, perhaps an infinite number. In this case, $\lambda(v)$ must be an associative and commutative operation, and make sense for infinite number of inputs.

In our analysis, we translate an SQL workflow into a summary dependency graph. The semantics of a summary dependency graph is the same as the semantics of the dependency graph resulting from its expansion. This semantics can be related to the semantics of the SQL workflow in a manner that shows their equivalence. We simplify the summary dependency graph, removing spurious dependencies, while changing the semantics of the graph only in a manner that still relates it to the SQL workflow. From the resulting graph, we can read out the actual dependencies of each output, including the actual computation, as well as the conditions of outputting them.

3.2.1.3 Translating SQL Workflows to Internal Representation. The translation of a query Q to a summary dependency graph works in syntax-directed manner. We first translate the database schema, resulting in a partial summary dependency graph (PSDG) consisting of only input nodes. Beside the PSDG, we also get a mapping from the attributes of tables to the nodes. This PSDG is given as the input to the translation of Q. The result is another PSDG, which will be post-processed to add the output nodes.

Let G be a PSDG, i.e. it may lack the output nodes. Consider a relation schema r with attributes $a_1, \ldots, a_n$. A *representation of r in G* is a mapping $R : \{\exists, a_1, \ldots, a_n\} \to V(G)$, such that $\mathsf{dim}(R(\exists)) = \mathsf{dim}(R(a_1)) = \cdots = \mathsf{dim}(R(a_n))$, the output type of each $R(a_i)$ matches with the type of a_i, and the output type of $R(\exists)$ is boolean. We write $\mathsf{dim}(R)$ for $\mathsf{dim}(R(\exists))$. A *representation* of a database schema *dbs in G* is a mapping from the contained relations into their representations in G.

Translating a database schema. The translation of a database schema *dbs* returns a PSDG G_{dbs}, as well as a representation R_{dbs} of *dbs* in it. These are the following:

- Let $t : r$ be a table declaration in *dbs*, where r is the relation schema $r(a_1 : D_1, \ldots, a_n : D_n; \mathsf{index}_r)$, with certain attributes belonging to the index. W.l.o.g. let $a_1, \ldots, a_h$ be the index attributes. The graph G will contain nodes $v^t_\exists$ and v^t_i for $1 \le i \le n$. The input dimension and the dimension of all nodes is $\mathcal{I} = \prod_{i=1}^{h} D_i$. All nodes are input nodes. During the execution, the instance $(x_1, \ldots, x_h)$ of the node v^t_i is supposed to carry the value of the attribute a_i in the row of the table t that corresponds to the index value $(a_1 = x_1, \ldots, a_h = x_h)$. The instance $(x_1, \ldots, x_h)$ of the node $v^t_\exists$ carries the value true iff the table t has a row with index value $(a_1 = x_1, \ldots, a_h = x_h)$.
- The representation R_{dbs} maps each table t to the mapping $\{\exists \mapsto v^t_\exists\} \cup \{a_i \mapsto v^t_i \mid 1 \le i \le |t|\}$.

Translating the query. The translation $\mathcal{G}$ of a query Q against a database with schema *dbs* takes as input a PSDG $G_\circ$ and a representation R_{dbs} of *dbs* in it. It returns a new PSDG $G_\bullet$ (which is obtained from $G_\circ$ by adding zero or more nodes to it) and a representation of $\mathbf{attr}(Q)$ in $G_\bullet$, where $\mathbf{attr}(Q)$ is the schema of the output relation of Q.

The translation $\mathcal{G}$ may call the translation $\mathcal{E}$ for expressions e. It takes as input a PSDG $G_\circ$ and a representation R of a relation schema in $G_\circ$. This relation schema must contain all attributes used by e. The translation $\mathcal{E}$ returns a new PSDG $G_\bullet$ and a node $v_e \in V(G_\bullet)$. The translation $\mathcal{E}$ works as follows.

- $\mathcal{E}[\![a]\!](G_\circ, R)$ returns $G_\circ$ and $R(a)$.
- $\mathcal{E}[\![\otimes(e_1, \ldots, e_k)]\!](G_\circ, R)$ calls $\mathcal{E}[\![e_1]\!], \ldots, \mathcal{E}[\![e_k]\!]$ one after another. Let the output of $\mathcal{E}[\![e_i]\!]$ be G_i and v_i. Then the inputs to $\mathcal{E}[\![e_i]\!]$ are G_{i-1} (with $G_0 \equiv G_\circ$) and R. After obtaining G_k, add a new node v to the graph. Its label is $\otimes$, and its dimension and input dimension are both $\mathsf{dim}(R)$. Also add arcs $\alpha_1, \ldots, \alpha_k$ to the graph, going from nodes $v_1, \ldots, v_k$ to the node v. For all i, the mapping $\overline{\delta}(\alpha_i)$ is equal to the identity map on $\mathsf{dim}(R)$. Return the modified graph G_k and the vertex v.

The translation $\mathcal{G}$ works as follows.

- $\mathcal{G}[\![t]\!](G_\circ, R_{dbs})$ returns $G_\circ$ and $R_{dbs}(t)$.
- $\mathcal{G}[\![Q_1 \times \cdots \times Q_k]\!](G_\circ, R_{dbs})$ calls $\mathcal{G}[\![Q_1]\!], \ldots, \mathcal{G}[\![Q_k]\!]$ one after another. Let the output of $\mathcal{G}[\![Q_i]\!]$ be G_i and R_i^Q. Then the inputs to $\mathcal{G}[\![Q_i]\!]$ are G_{i-1} (with $G_0 \equiv G_\circ$) and R_{dbs}. After obtaining G_k and $R_1^Q, \ldots, R_k^Q$, we add the following nodes and arcs to G_k:
  - Let $\mathcal{I} = \prod_{i=1}^k \mathsf{dim}(R_i^Q)$.
  - Add a node $v_\exists$. The label of this node is "&" (boolean conjunction). Its dimension and input dimension are both $\mathcal{I}$.
  - For each $i \in \{1, \ldots, k\}$ add an arc $\alpha_{\exists,i}$ from the node $R_i^Q(\exists)$ to $v_\exists$. The mapping $\overline{\delta}(\alpha_{\exists,i})$ is the canonical projection from $\mathcal{I}$ to its i-th component $\mathsf{dim}(R_i^Q)$.
  - For each $i \in \{1, \ldots, k\}$ and each attribute $a_j \in \mathbf{attr}(Q_i)$ add a node $v_{i,j}$. The label of this node is "ID" (the identity mapping). Its dimension and input dimension are both $\mathcal{I}$.
  - Also, add an arc $\alpha_{i,j}$ from $R_i^Q(a_j)$ to $v_{i,j}$. The mapping $\overline{\delta}(\alpha_{i,j})$ is the canonical projection from $\mathcal{I}$ to its i-th component $\mathsf{dim}(R_i^Q)$.

  Let the output PSDG $G_\bullet$ be the modified graph G_k. The output representation R maps $\exists$ to $v_\exists$ and the attribute a_j in $\mathbf{attr}(Q_i)$ to $v_{i,j}$.
- $\mathcal{G}[\![[Q]_{a \to a'}]\!](G_\circ, R_{dbs})$ runs $(G_\bullet, R) = \mathcal{G}[\![Q]\!](G_\circ, R_{dbs})$. It returns $G_\bullet$ and $R[a' \mapsto R(a)]$.
- $\mathcal{G}[\![\sigma(Q; e)]\!](G_\circ, R_{dbs})$ runs $(G', R) = \mathcal{G}[\![Q]\!](G_\circ, R_{dbs})$ and $(G'', v_?) = \mathcal{E}[\![e]\!](G', R)$. It adds a node $v_\exists$ to G''. The label of this node is "&" and both its dimension and input dimension are $\mathsf{dim}(R)$. The node $v_\exists$ has two inputs, from $R(\exists)$ and from $v_?$. The $\overline{\delta}(\cdot)$-mappings of both respective arcs are the identity mappings over $\mathsf{dim}(R)$. Let $G_\bullet$ be the modified graph G''. The translation returns $G_\bullet$ and $R[\exists \mapsto v_\exists]$.
- $\mathcal{G}[\![\pi_{a_1,\ldots,a_k}(Q)]\!](G_\circ, R_{dbs})$ runs $(G_\bullet, R) = \mathcal{G}[\![Q]\!](G_\circ, R_{dbs})$. It returns $G_\bullet$ and R restricted to $\{\exists, a_1, \ldots, a_k\}$.
- $\mathcal{G}[\![\mathsf{col}_{a \leftarrow e}(Q)]\!](G_\circ, R_{dbs})$ runs $(G', R) = \mathcal{G}[\![Q]\!](G_\circ, R_{dbs})$ and $(G_\bullet, v_e) = \mathcal{E}[\![e]\!](G', R)$. It returns $G_\bullet$ and $R[a \mapsto v_e]$.
- $\mathcal{G}[\![\mathsf{let}\ t = Q_1\ \mathsf{in}\ Q_2]\!](G_\circ, R_{dbs})$ runs $(G', R_0) = \mathcal{G}[\![Q_1]\!](G_\circ, R_{dbs})$, followed by $(G_\bullet, R) = \mathcal{G}[\![Q_2]\!](G', R_{dbs}[t \mapsto R_0])$. It returns $G_\bullet$ and R.

Adding output nodes. Let the query Q be translated by calling $\mathcal{G}[\![Q]\!]$ on the translation of the database schema. The result of $\mathcal{G}[\![Q]\!]$ is a PSDG G and a representation R of $\mathbf{attr}(Q)$ in G. We add the following nodes and arcs to G:

- For each $a_i \in \mathbf{attr}(Q)$, add nodes v_i and v_i^O. For both of them, their dimension and input dimension are equal to $\mathsf{dim}(R)$. Node v_i is an internal node, while v_i^O is an output node. There is an arc from v_i to v_i^O; its $\overline{\delta}(\cdot)$-mapping is the identity mapping on $\mathsf{dim}(R)$. There are two arcs into v_i, first from $R(\exists)$ and second from $R(a_i)$. Their $\overline{\delta}(\cdot)$-mappings are also the identity mappings on $\mathsf{dim}(R)$. The operation of v_i is named "Output". The semantics of an "Output" operation is to return the second argument, if the first argument is true, and to return NULL otherwise.

3.2.1.4 Example. We consider a part of the Aid Distribution example of the Enterprise CRT. The fragment below declares the tables `port` and `ship`, and an SQL workflow that

- selects the ports that a ship with the name `shipname` can reach before `deadline`;
- filters out the ports that the ship cannot fit into, returning only the identities of suitable ports;
- maps the port identities to their names.

```
CREATE TABLE ship (
  ship_id bigserial PRIMARY KEY,
  name text UNIQUE NOT NULL,
  cargo bigint,
  latitude DOUBLE PRECISION,
  longitude DOUBLE PRECISION,
  length bigint,
  draft bigint,
  maxspeed bigint
);

CREATE TABLE port (
  port_id bigserial PRIMARY KEY,
  name text NOT NULL,
  latitude DOUBLE PRECISION,
  longitude DOUBLE PRECISION,
  offloadcapacity bigint,
  offloadtime bigint,
  harbordepth bigint,
  available boolean
);

create function compute_reachable_ports(deadline BIGINT, shipname TEXT)
  returns TABLE (port_id BIGINT, arrival BIGINT) as
$$
  select port.port_id, distance(ship.longitude, ship.latitude,
    port.longitude, port.latitude) / ship.maxspeed as arrival
  from port, ship
  where arrival <= deadline
        and ship.name = shipname;
$$
language SQL;

SELECT * INTO reachable_ports
FROM compute_reachable_ports(:deadline, :shipname);

create function compute_feasible_ports(shipname TEXT)
  returns TABLE (port_id BIGINT) as
$$
  select port.port_id
  from reachable_ports, port, ship
  where reachable_ports.port_id = port.port_id
    and port.available = true
    and port.harbordepth >= ship.draft
    and port.offloadcapacity >= ship.cargo
    and ship.name = shipname;
$$
language SQL;

select * into feasible_ports
from compute_feasible_ports(:shipname);

SELECT port.port_id, port.name
FROM feasible_ports, port
WHERE feasible_ports.port_id = port.port_id;
```

In our analysis, we cannot handle parametric queries. Hence we add a table `parameters`, which will be joined with other tables in each SQL-statement of the workflow.

```
CREATE TABLE parameters (
  pm_idx unit PRIMARY KEY,
  shipname text NOT NULL,
  deadline bigint
);
```

As mentioned above, each table must have an index, hence we give the `parameters`-table the index attribute `pm_idx`. Notably, the type of `pm_idx` is `unit`, hence there may be at most one row in this table.

The SQL workflow is translated into the query language given in Fig. 10. The translation of that query, using the procedure given in Sec. 3.2.1.3 gives us the SDG depicted in Fig. 12, after we have removed all dead nodes from it, and fused its identity operations.

Let us explain the details of the SDG and its visualization in Fig. 12. Rectangles with square corners depict the nodes v of the graph. Top left indicates the operation $\lambda(v)$ of the node, while top right gives a unique identifier of the node. The rest of the depiction shows $\mathsf{dim}(v)$ (which in this example is always equal to $\overrightarrow{\mathsf{dim}}(v)$). Namely, in this example, $\mathsf{dim}(v)$ is the product of some of the sets from $\mathcal{S}$, which has elements "integer" and "unit". The middle column lists the components of the product, the right column lists the names of index columns which contributed this component of the product, and the left column numbers the rows.

An arc α, where $\overline{\delta}(\alpha)$ is the identity mapping, is depicted as an arrow from the source to the target node. If $\overline{\delta}(\alpha)$ is not identity, then it is depicted in the middle of that edge, inside a rectangle with rounded corners. The right part of that rectangle lists $\mathsf{dim}(\mathsf{t}\alpha)$, and the left part $\mathsf{dim}(\mathsf{s}\alpha)$, with the numbers at the source indicating the sequence number of the target dimension that they match. For certain nodes, where the order of incoming arcs matters, the arcs have labels in the middle (or in the top part of the rounded rectangle). For the node computing the geographic distance between two geographic coordinates, and for other nodes doing arithmetic and relational operations, the label states the sequence number of the argument. For an output node, whose arguments are the value to be output and the condition under which it is output, the arc from the output value is unlabeled. The arc from the output condition is labeled with "C".

The input nodes in Fig. 12 are labeled. Namely, the node $v^t_{\exists}$ is labeled with "Exists t". The node v^t_i for an attribute a_i belonging to the index of the table t is labeled with "TakeDim a_i". If a_i does not belong to the index, then the node is labeled "Input $t.a_i$".

We can recognize the workflow from the SDG in Fig. 12. There are two outputs, `port.port_id` and `port.name`. As they are taken from the table `port` at the last SQL query, they are directly passed to the output nodes. The rest of the graph is involved in computing the condition when these outputs can be released (node 1244). It takes two inputs, one comparing the `port_id` attributes of tables `feasible_ports` and `port` (node 1241) and the other one expressing the condition that the rows in both tables exist (node 1238). The existence of the row in table `port` is expressed straightforwardly (node 19). A more involved condition is in place for checking that the row in the table `feasible_ports` exists (node 177). Again, this condition consists of two parts, one expressing the **`WHERE`**-clause of the SQL statement defining the `feasible_ports` table (node 171) and the other one expressing the requirement for relevant rows in all tables joined at this SQL statement (`reachable_ports`, `port`, `ship`, but also `parameters`, as the input parameters are used in this statement) to exist (node 154). We can continue our description in the same manner. We see that there is a lot of redundancy, and a lot of it is due to the definition of the workflow which see the joining of the same tables over and over again.

3.2.1.5 Simplifications on Internal Representation. We have implemented a number of simplifications of SDG, both structural and semantical. The current set of simplifications is selected based on the needs of the (full) Aid Distribution scenario. A simplification operation, applied to a certain node or a group of nodes, checks whether the local context of these nodes matches some pattern. If it does, then these nodes are replaced with some other nodes that have the same effect semantically (or an effect that is similar in the view of our task to find which inputs end up where, how, and when), but have simpler structure. One simplification may enable others. We thus run the simplifications in the order that seems to make the most sense; some simplifications (e.g. the removal of dead nodes) is run many times. In the following, we will describe some simplifications that our analyzer currently runs.

Removal of dead nodes. A node that has no descendants may be removed, unless it is an output node. Running this removal many times, we will remove all nodes that are not backwards reachable from any output node.

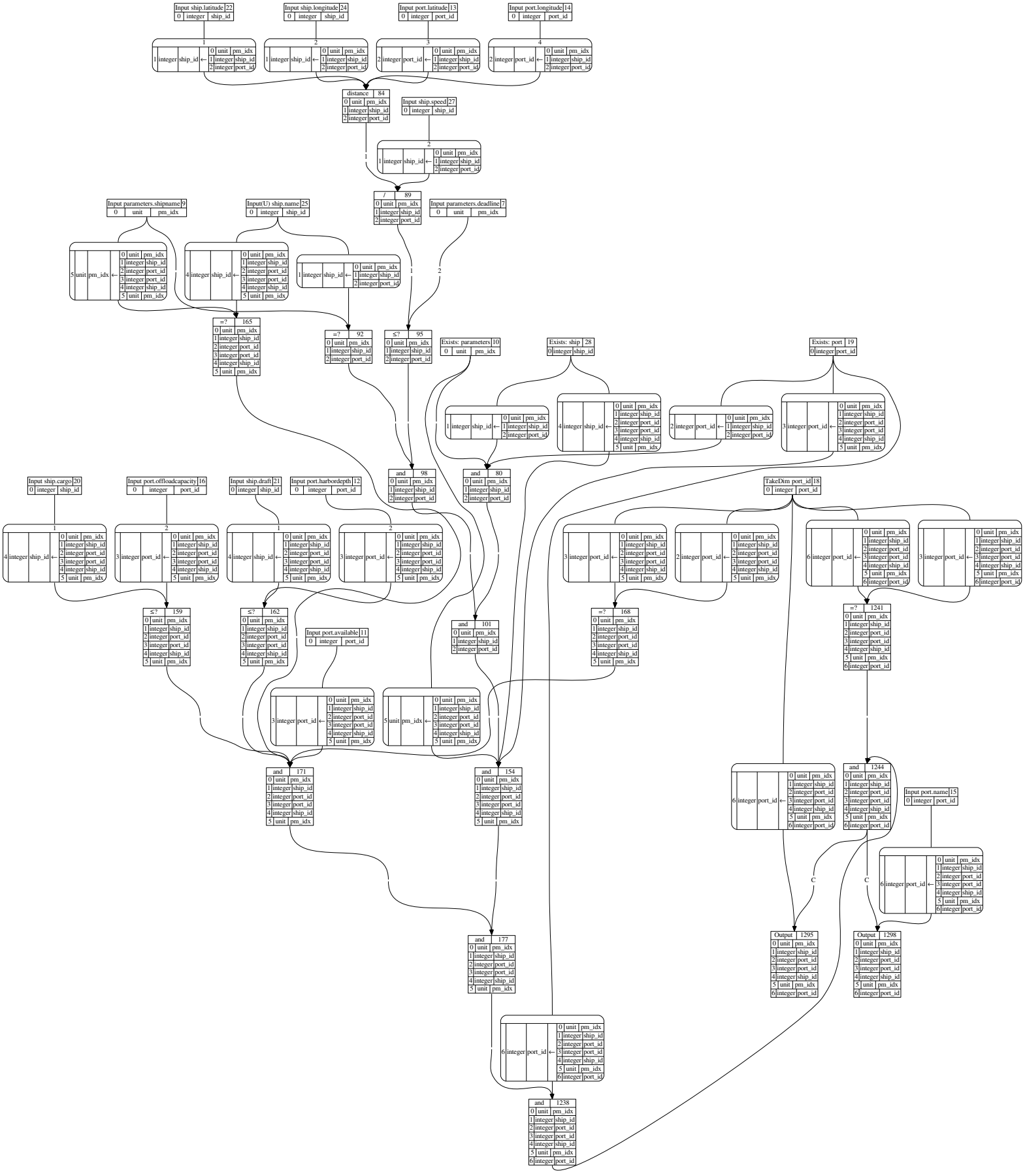


Figure 12: Initial Summary Dependency Graph for Our Example (Without Dead or ID Nodes)

Folding of identity operations. An ID node can be cut out of paths: if v is an ID node and α is the arc leading to it, and β is any arc with the source v, then β may be replaced with the arc $\beta \circ \alpha$: we define $\mathsf{s}(\beta \circ \alpha) = \mathsf{s}(\alpha)$, $\mathsf{t}(\beta \circ \alpha) = \mathsf{t}(\beta)$ and $\overline{\delta}(\beta \circ \alpha) = \overline{\delta}(\alpha) \circ \overline{\delta}(\beta)$, assuming that $\delta(v)$ is the identity mapping (which is always the case in the SDGs that we construct). After all arcs leaving v have been replaced, v is dead and can be removed by the previous simplification.

Folding the "&"-nodes. If v and v' are both conjunctions, and there is an arc α from v' to v, then we add arcs from all predecessors of v' to v (with the correct $\overline{\delta}(\cdot)$-mapping) and remove the arc α. If there were no other arcs leaving v', then it is dead.

Joining nodes with identical computation. If there are two nodes with the same operation and the same inputs, then these nodes can be turned to a single node. In our SDGs, the recognition of these nodes is complicated by the need to determine if a suitable isomorphism between their dimensions exists.

Reducing the dimension of a node. In our SDG-s, the dimensions of nodes are products of elements of $\mathcal{S}$. If for some node v in SDG, the predecessors of the nodes corresponding to v in the infinite dependency graph do not depend on some component of the elements in $\mathsf{dim}(v)$, then this component may be removed from $\mathsf{dim}(v)$. In Fig. 12, there are several nodes to which this simplification would apply, starting from node 84, where the `pm_idx` component of its dimension is superfluous.

Joining components of dimensions. Consider node 168 in Fig. 12. It compares `port.port_id` and `reachable_ports.port_id` in the function that defines the table `feasible_ports`. We see that node 168 can return `true` only if the components no. 2 and 3 of its dimension are the same. We can identify them and reduce the dimension of the node. This reduction works differently from the previous simplification, and has to be propagated along the SDG in both directions.

Uniqueness constraints can also be used to join components of dimensions. In Fig. 12, we can find out that components no. 1 and no. 4 (let us denote them x_1 and x_4) of the dimension of node 177 must be equal, if this node outputs `true`. There reason is, that node 92, which is an ancestor of node 177, compares `ship.name` for the value x_1 of `ship_id` against the parameter `shipname`. At the same time, node 165, which is also an ancestor of node 177, compares `ship.name` for the value x_4 of `ship_id` against the same parameter. Both comparisions can return `true` only if `ship.name` is the same for the values x_1 and x_4 of the index `ship_id`. But as `ship.name` is unique, x_1 and x_4 must be the same.

Arithmetic simplifications. A conjunction with a single input, or a sum with a single input can be turned to an ID node. A conjunction with a FALSE-input can be turned to FALSE-node (with no inputs). A `COALESCE`-operation (occurs in the last step of the Aid Distribution scenario) can also be simplified if we know that some of its arguments certainly are, or certainly are not NULL.

The application of these simplifications to the SDG depicted in Fig. 12 results in the graph depicted in Fig. 13. It shows much more clearly, when the values are output.

3.2.1.6 Interpreting the Result. The dependencies and conditions are depicted in our final SDG, but they are not given in terms of certain rows existing or not existing in the tables of the database. To present the outcome, we have to map from the product of elements of $\mathcal{S}$ back into tables. In the graph shown in Fig. 13 it is easy to see that `ship_id` corresponds to a row in the table `ship` and `port_id` to a row in the table `port`. We can read that the attributes `port_id` and `name` of some row R in the table `port` are released, IF there exists a row R' in the table `ship`, SUCH THAT

- R'.`cargo` is less that R.`offloadcapacity`;
- R.`available` is true;
- The distance between the ship R' and the port R, divided by R'.`speed`, is less than the parameter `deadline`;
- R'.`draft` is less than R.`harbordepth`;

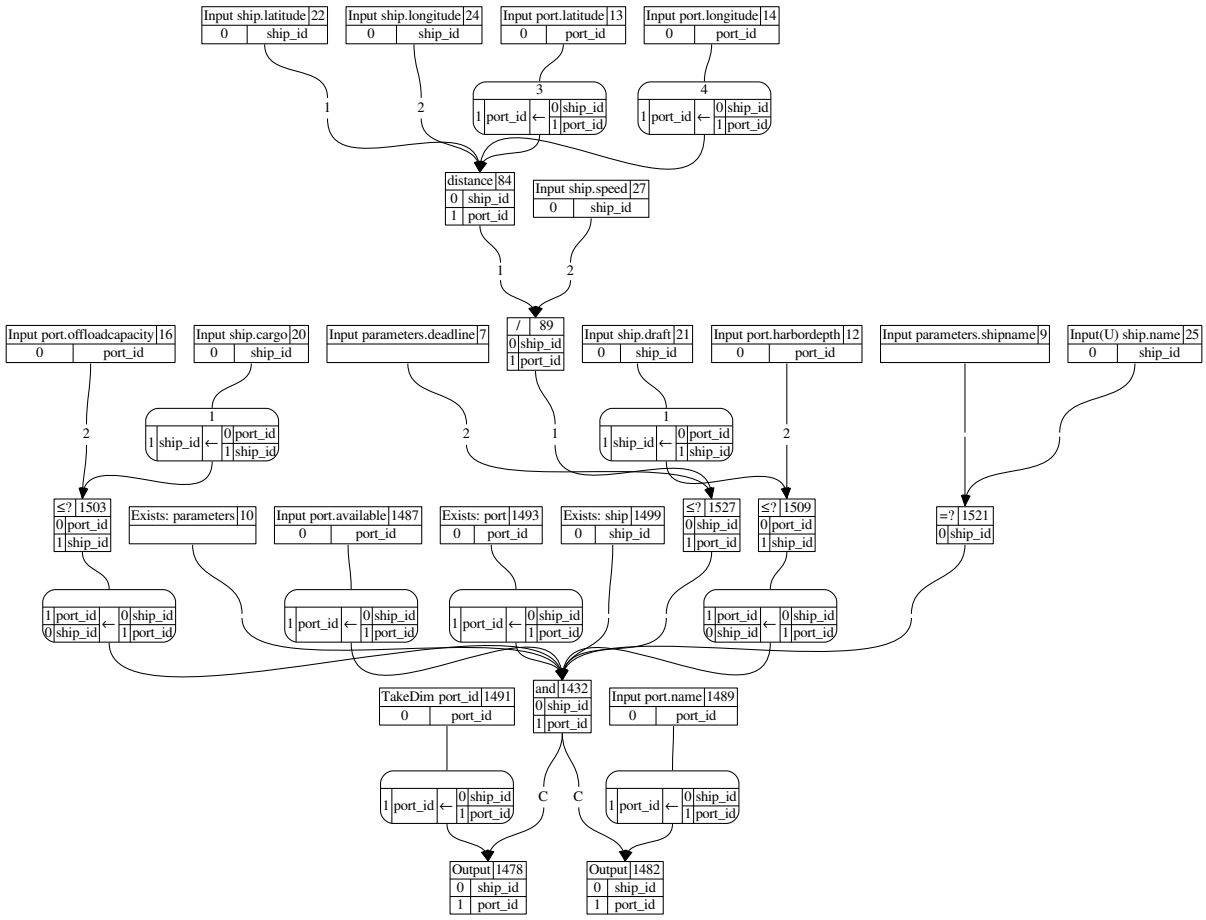


Figure 13: Final Summary Dependency Graph for Our Example

$$Q ::= \ldots \mid Q_1 \cup Q_2 \mid Q_1 \cap Q_2 \qquad (a)$$

$$Q ::= \ldots \mid Q_1 \ltimes_e Q_2 \qquad (b)$$

$$Q ::= \ldots \mid \mathsf{group}^{a_1,\ldots,a_k}_{(a'_1 \bigotimes_1),\ldots,(a'_l,\bigotimes_l)}(Q) \qquad (c)$$

Figure 14: Syntax of Additional Operations in Queries

- R'.`name` is equal to the parameter `shipname`.

In general, the translation of attribute names back into table rows is less straightforward, if some table has the index consisting of more than one component, or if several rows from the same table are involved. In general, we track each dimension component from the place it is introduced (either at some Output node, or at a node that has larger input dimension than its dimension) to input nodes. The input nodes refer to a table, and to a particular row. From them, we find the involved rows and the equalities between their index attributes.

3.2.1.7 Other Operations in Queries. The query language given in Fig. 10 allows for workflows with simple SELECT ... FROM ... WHERE ... queries. However, the Aid Distribution scenario contains queries with more complex operations. Hence we will introduce them now, discussing their translation into SDGs and any optimizations that are specific to them.

3.2.1.8 Set Operations. In certain workflows we have to perform unions or intersections or other set operations with the results of the queries in the workflow. We thus extend our query language as shown in Fig. 14a. In these set operations, we require Q_1 and Q_2 to have the same schema. In this case, the

meaning of these operations is obvious. We extend the definition of $\mathcal{G}$ to translate these operations into SDG as follows:

- $\mathcal{G}[\![Q_1 \cup Q_2]\!](G_\circ, R_{dbs})$ runs $(G', R') = \mathcal{G}[\![Q_1]\!](G_\circ, R_{dbs})$ and $(G'', R'') = \mathcal{G}[\![Q_2]\!](G', R_{dbs})$. For each attribute $a \in \mathbf{attr}(Q_1) = \mathbf{attr}(Q_2)$ it will then add a node v_a to G'', with the operation "ID" and its dimension and input dimension both being equal to $\mathsf{dim}(R') + \mathsf{dim}(R'')$. The mapping $\delta(v_a)$ is the identity mapping. The node v_a has a single incoming arc α_a, which has *two* sources — $R'(a)$ and $R''(a)$. The mapping $\overline{\delta}(\alpha_a)$ is the identity mapping from $\overrightarrow{\mathsf{dim}}(v_a)$ to $\mathsf{dim}(R'(a)) + \mathsf{dim}(R''(a))$.

  We also add a node $v_\exists$ to the graph G'' with the same dimension, input dimension and $\delta(\cdot)$-mapping as described in the previous paragraph. The operation in this node is again "ID" (boolean disjunction), and it again has a single incoming arc $\alpha_\exists$ with two sources: $R'(\exists)$ and $R''(\exists)$, with the mapping $\overline{\delta}(()\alpha_\exists)$ again being the identity map.

  Let the output PDSG $G_\bullet$ be the graph G'' with the added nodes and arcs. The output representation R maps $\exists$ to $v_\exists$ and each attribute a to v_a.

- $\mathcal{G}[\![Q_1 \cap Q_2]\!](G_\circ, R_{dbs})$ runs $(G', R') = \mathcal{G}[\![\sigma(Q_1 \times [Q_2]_{a:\mathbf{attr}(Q_2)\to a'}; \bigwedge_{a\in\mathbf{attr}(Q_1)} a = a')]\!](G_\circ, R_{dbs})$ first, while also keeping the representation R_1 that was produced while $\mathcal{G}[\![Q_1]\!](G_\circ, R_{dbs})$ was run as a subroutine. Here the write-up $[Q_2]_{a:\mathbf{attr}(Q_2)\to a'}$ denotes that we have renamed all attributes a of Q_2 into their primed versions.

  We add to G' a node $v_\exists$ with the operation "$\bigvee$" (boolean disjunction). We let $\mathsf{dim}(v_\exists) = \mathsf{dim}(R_1)$ and $\overrightarrow{\mathsf{dim}}(v_\exists) = \mathsf{dim}(R')$. Recall that $\mathsf{dim}(R')$ is equal to the Cartesian product of $\mathsf{dim}(R_1)$ and the dimension of the nodes resulting from the translation of the query Q_2. The mapping $\delta(v_\exists)$ is the natural projection to the first component of this product.

  As $\mathsf{dim}(v_\exists) \neq \overrightarrow{\mathsf{dim}}(v_\exists)$, this node may have a single incoming arc. This arc comes from the node $R'(\exists)$, its $\overline{\delta}(\cdot)$-mapping is the identity mapping.

  We return the graph G' with the extra node and arc. As the output representation, we return $R_1[\exists \mapsto v_\exists]$.

In these translations we first see edges with several sources, as well as nodes whose dimension is different from the input dimension. The disjunction node effectively indicates if there is any row in $Q_1 \times Q_2$ where the Q_2-part is equal to the attributes that the row in Q_1 has.

There is an optimization related specifically to the edges introduced by the translation of unions into SDGs.

Splitting the dimensions that are sums. Let v be a node in the SDG, its dimension can be expressed as a polynomial over $\mathcal{S}$. If this polynomial has k monomials, with $k > 1$, then we replace v with nodes $v_1, \ldots, v_k$, whose dimensions are equal to the monomials of $\mathsf{dim}(v)$. We do this replacement in parallel for all nodes in the SDG. The edges can be adjusted accordingly. This change simplifies the logic of following simplifications.

Outer joins. Our query language allows to express inner joins of datasets — $\sigma(Q_1 \times Q_2; e)$ is the inner join of Q_1 and Q_2 over the boolean expression e. A left outer join additionally includes all rows in Q_1 that have no matching row in Q_2 according to e; the Q_2-attributes of such rows in the result of the query are set to NULLs. Right outer join is defined symmetrically. To express outer joins, we have added to our query language a construction depicted in Fig. 14b. The meaning of this construction is the *difference* between the left outer join and the inner join of Q_1 and Q_2, with e serving as the condition.

The translation $\mathcal{G}[\![Q_1 \ltimes_e Q_2]\!](G_\circ, R_{dbs})$ is the following. We execute

$$(G', R_2) = \mathcal{G}[\![Q_1 \times Q_2]\!](G_\circ, R_{dbs})$$
$$(G'', v_e) = \mathcal{E}[\![e]\!](G_2, R_2) \ .$$

We also keep the representation R_1 that was produced when $\mathcal{G}[\![Q_1]\!](G_\circ, R_{dbs})$ was run as a subroutine. After that, we add the following nodes and arcs to G''.

- Node v_1, operation "&", with dimension and input dimension equal to $\mathsf{dim}(R_2)$. Its inputs are v_e and $R_2(\exists)$.
- Node v_2, operation "$\bigvee$". Its dimension is equal to $\mathsf{dim}(R_1)$ and its input dimension to $\mathsf{dim}(R_2)$. The mapping $\delta(v_2)$ is the natural projection from the second to the first. The input to v_2 is the node v_1.
- Node v_3, operation "NOT". Its dimension and input dimension are equal to $\mathsf{dim}(R_1)$. Its input is the node v_2.
- Node v_4, operation "&". Its inputs are v_3 and $R_1(\exists)$.

For all arcs described above, their $\overline{\delta}(\cdot)$-mapping is the identity mapping. The translation returns the PSDG G'' together with added nodes and arcs. As the output representation, it returns $R_1[\exists \mapsto v_4]$.

The operation $Q_1 \ltimes_e Q_2$ can be used to construct left and right outer joins of Q_1 and Q_2 (the latter by swapping the position of Q_1 and Q_2), as well as their full outer join. Indeed, if $\mathbf{attr}(Q_2) = \{a'_1, \ldots, a'_k\}$ then their left outer join is expressed as

$$\sigma(Q_1 \times Q_2; e) \cup \mathsf{col}_{a'_1 \leftarrow NULL}(\mathsf{col}_{a'_2 \leftarrow NULL}(\cdots \mathsf{col}_{a'_k \leftarrow NULL}(Q_1 \ltimes_e Q_2) \cdots))$$

and the full outer join would see $Q_2 \ltimes_e Q_1$, expanded with NULL columns, additionally added to it.

Aggregations. A SQL statement may have the `GROUP BY` component. The Aid Distribution scenario does not explicitly contain such components, but we need them in Sec. 3.2.1.8 and hence introduce them into our language as depicted in Fig. 14c. The operation depicted there requires $\mathbf{attr}(Q)$ to contain the attributes $a_1, \ldots, a_k$ and $a'_1, \ldots, a'_l$. These attributes are exactly those that are left after the grouping query, with $\{a_1, \ldots, a_k\}$ forming the index of the result. To formally describe the semantics of the grouping construct, let T be the dataset that is the outcome of the query Q and let T' be the expected outcome of the grouping. Then

- T has attributes $a_1, \ldots, a_k, a'_1, \ldots, a'_l$;
- Let $(x_1, \ldots, x_k)$ be a possible value for the attributes $a_1, \ldots, a_k$. If T contains a row where the values of the attributes $a_1, \ldots, a_k$ are $x_1, \ldots, x_k$, then T' contains exactly one row where these attributes have these values. Let us denote this row $T'[x_1, \ldots, x_k]$. If there are no rows in T where the values of the attributes $a_1, \ldots, a_k$ are $x_1, \ldots, x_k$, then T' also has no such rows.
- Let the row $T'[x_1, \ldots, x_k]$ exist and let $R_1, \ldots, R_n$ be all rows in T, where the attributes $a_1, \ldots, a_k$ have the values $x_1, \ldots, x_k$. Then, for all $j \in \{1, \ldots, l\}$, the value of the attribute a'_j in the row $T[x_1, \ldots, x_k]$ is computed as follows:

$$T[x_1, \ldots, x_k](a'_j) = \bigotimes_{i=1}^{n}{}_j \, R_i.a'_j \ .$$

The translation $\mathcal{G}[\![\mathsf{group}^{a_1,\ldots,a_k}_{(a'_1 \otimes_1),\ldots,(a'_l,\otimes_l)}(Q)]\!](G_\circ, R_{dbs})$ works as follows. It first executes $(G', R') = \mathcal{G}[\![Q]\!](G_\circ, R_{dbs})$. It will determine the types $D_1, \ldots, D_k$ of the attributes $a_1, \ldots, a_k$ of Q. These types must be elements of $\mathcal{S}$. The following nodes and arcs are then added to G':

- Nodes $v_1^{\mathrm{TD}}, \ldots, v_k^{\mathrm{TD}}$. These are input nodes of the SDG. The dimension of v_i^{TD} is D_i. In the infinite dependency graph, a node v corresponding to the value $x \in D_i$ and the node v_i^{TD}, is expected to carry the value x. Let $\mathcal{I} = D_1 \times \cdots \times D_k$.
- Nodes $v_1^{=}, \ldots, v_k^{=}$. The operation of these nodes is "=" (equality check). The dimension and input dimension of these nodes is $\mathsf{dim}(R') \times \mathcal{I}$. The node $v_i^{=}$ has two inputs: v_i^{TD} and $R'(a_i)$. The $\overline{\delta}(\cdot)$-mappings for the arcs connecting these nodes are the natural projections.
- Node $v^{=}$. The operation of this node is "&". Its dimension and input dimension are both $\mathsf{dim}(R') \times \mathcal{I}$. Its inputs are the nodes $v_1^{=}, \ldots, v_k^{=}$.

- Node $v_\exists$. The operation of this node is "$\bigvee$". Its dimension is $\mathcal{I}$ and its input dimension is $\mathsf{dim}(R') \times \mathcal{I}$. The mapping $\delta(w_\exists)$ is the natural projection. Node $v_\exists$ receives its input from $v^=$.
- Nodes $v_1^f, \ldots, v_l^f$. The operation of these node is "Output" (see the end of Sec. 3.2.1.3). Their dimension and input dimension are $\mathsf{dim}(R') \times \mathcal{I}$. The inputs of the node v_j^f are $v^=$ (for the first, "conditioning" argument) and $R'(a'_j)$ (for the second, "value" argument). The $\overline{\delta}(\cdot)$-mapping for the arc connecting to the first input is the identity mapping, while for the arc connecting to the second input is the natural projection from $\mathsf{dim}(R') \times \mathcal{I}$ to $\mathsf{dim}(R')$.
- Nodes $v_1^\otimes, \ldots, v_l^\otimes$. The operation of the node $v_j^\otimes$ is "$\bigotimes_j$". The dimension of $v_j^\otimes$ is $\mathcal{I}$, while its input dimension is $\mathsf{dim}(R') \times \mathcal{I}$. The mapping $\delta(v_j^\otimes)$ is the natural projection. The input to the node $v_j^\otimes$ is the node v_j^f.

We see that the expansions of the nodes $v_j^\otimes$ in the infinite dependency graph perform the actual aggregations of the values of the dataset resulting from the query Q. We have implicitly assumed that the NULL-values among the inputs of the operations $\bigotimes_j$ do not change their output value.

The translation returns the graph G' together with the added nodes and arcs. The output representation R is the following:

- $R(\exists) = w_\exists$;
- $R(a_i) = v_i^{\mathrm{TD}}$ for the attributes $a_1, \ldots, a_k$;
- $R(a'_j) = v_j^\otimes$ for the attributes $a'_1, \ldots, a'_l$.

Sorting and sequence numbers. In our analysis, consider the outputs of the queries to be *sets* of rows. As sorting the result of a query does not change that set, we have in general ignored it. However, SQL dialects contain the function `row_number()` which returns the sequence number of the current row in the dataset. More generally, the row number generation can be done after the dataset has been partitioned according to the values of some other column(s). The row numbers of sorted datasets have been used in the Aid Distribution scenario in significant manner.

We model the generation of row numbers in such way not through a separate construction in our query language, but by defining specific operations and aggregation functions, such that their combination produces the necessary sequence numbers. Namely, we define an aggregation operation BAG. It receives as its input any number of pairs, such that the first components of these pairs are non-repeating. We assume that the number of non-NULL inputs is finite. The operation returns the list of these pairs, sorted by their second component. We treat this list as a single value, i.e. it may be passed over the arcs of a dependency graph.

We also define the (normal) operation SeqNo. It receives a list of pairs L and a value x. If x is among the first components of the pairs in L, then SeqNo returns the position in which the pair containing x occurs in L. If there are no such pairs, then SeqNo returns NULL. The addition of a column a_{sn} to the result of the query Q, which contains the sequence number of the row when the result of Q is sorted by the value of the column a_{ord}, and the sequence numbers are defined separately for each possible value of the column a_{prt}, is expressed as follows:

$$\pi_{\mathbf{attr}(Q)}(\mathsf{col}_{a_{\mathrm{sn}} \leftarrow \mathrm{SeqNo}(a_{\mathrm{pair}}, a_{\mathrm{prt}})}(\sigma(Q \times \mathsf{group}^{a'_{\mathrm{prt}}}_{(a_{\mathrm{pair}}, \mathrm{BAG})}(\mathsf{col}_{a_{\mathrm{pair}} \leftarrow \langle a'_{\mathrm{prt}}, a_{\mathrm{ord}} \rangle}([Q]_{a_{\mathrm{prt}} \to a'_{\mathrm{prt}}})); a_{\mathrm{prt}} = a'_{\mathrm{prt}}))) \ .$$

3.2.2 Analysis of Collaborative SQL Workflows. In processes with many parties, the SQL workflow is executed collaboratively by them, with different parties running different SQL queries in the workflow, constructing the results of these queries, and sending the tables to each other. Thus we define a *SQL collaboration workflow* as a collaboration model in the standard BPMN notation in which each pool corresponds to a SQL workflow, with incoming and outgoing messages carrying tables.

Each task in a SQL workflow is a query over a set of input tables from a database (the inputs of the tasks) and produces new tables (the outputs of the task), which may later be used by subsequent tasks in the workflow. The table (or set of tables) that are taken as input by the first SQL queries in the

collaboration workflow are the inputs of the workflow. Conversely, the tables produced by the last SQL queries in the collaboration workflow are the final output(s). We call the tables produced by intermediate tasks in the collaboration workflow intermediate outputs.

As a running example, Fig. 15 presents an example SQL collaboration workflow adapted from an Aid Distribution scenario, developed by the Enterprise CRT, in which a country facing a catastrophe, requests aid from the international community. The situation requires distributing goods to the population via maritime transportation. Henceforth, a SQL collaboration workflow is executed to identify ships in nearby locations and to allocate berths to ships, such that ships can move people and goods from/to the requesting country.

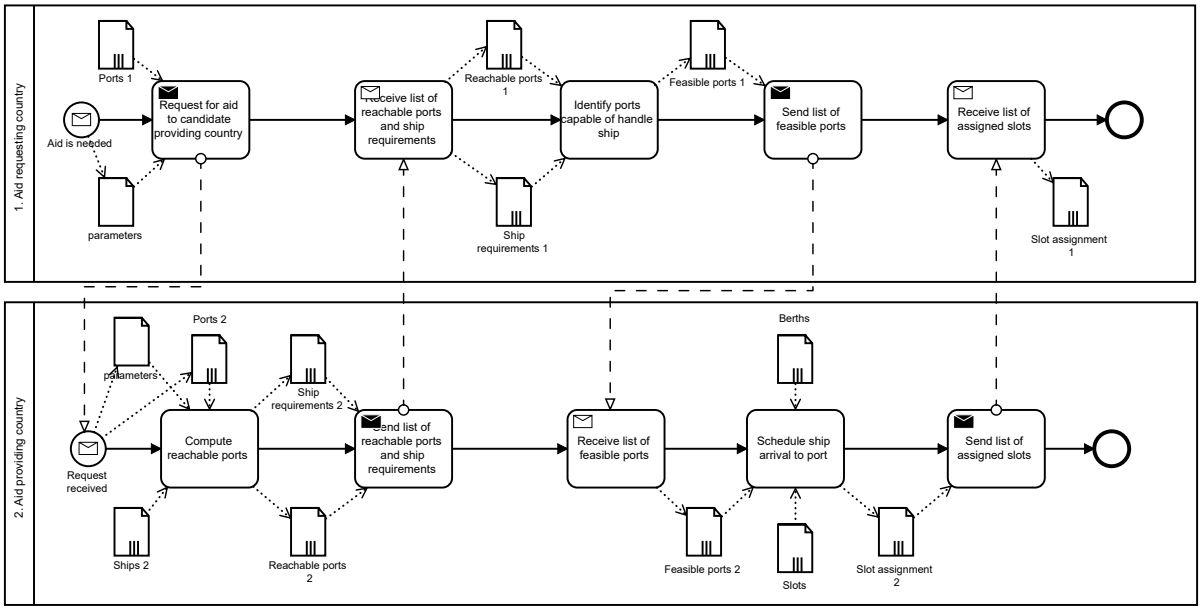


Figure 15: *Aid Distribution* Scenario Captured as BPMN Collaboration Model

In the model, it is assumed that each participant maintains a local, private database, and that data is exchanged between participants via message passing. Naturally, parts of the information used in this collaboration are confidential (e.g. ship location and capacities) and the countries involved in the collaboration seek to minimize the (private) data they expose to other stakeholders. Accordingly, an analyst needs to determine: (i) who gets access to which input tables during the performance of the process? (ii) what information (e.g. table columns or functions over columns) are disclosed? and (iii) under what conditions this disclosure occurs? Our disclosure analysis technique supports this task by determining what information is disclosed via each intermediate and final output of the collaboration workflow, and under which conditions (i.e. for which table rows) this disclosure occurs.

3.2.2.1 SQL Collaboration Workflows. For the disclosure analysis, we assume that the overall computation is distributed over a set of nodes, each of them under the responsibility of one of the participants on the collaboration. In that context, it is assumed that each node maintains a local database and runs a local workflow, which is composed of SQL statements that manipulates the local database. Moreover, each one of the local workflows shares part of the local database with other nodes by message passing. Therefore, we assume that each task in a collaboration workflow can be a data processing task or one that serves to sending/receiving messages to other partners.

One example of the former is the task "Compute reachable ports" in the running example. Listing 3.1 presents the SQL script associated with such a task.

Listing 3.1: SQL Script Associated with Task "Compute reachable ports"

```
1  create function earliest_arrival(
2    ship_latitude double, ship_longitude double,
```

```
3     port_latitude double, port_longitude double,
4     max_speed bigint) returns bigint as
5   $$
6   select ceil((point(ship_latitude, ship_longitude)
7               <@> point(port_latitude, port_longitude)) / max_speed)::bigint
8   $$
9   language SQL immutable returns null on null input;
10
11  select port.port_id as port_id,
12    earliest_arrival(ship.longitude, ship.latitude, port.longitude,
13                     port.latitude, ship.maxspeed) as arrival
14  into reachable_ports2
15  from ports2 as port, ships2 as ship, parameters as p
16  where earliest_arrival(ship.longitude, ship.latitude,
17            port.longitude, port.latitude, ship.maxspeed) <= p.deadline;
18
19  select port.port_id as port_id,
20         ship.draft as draft,
21         ship.cargo as cargo
22  into ship_requirements2
23  from ports2 as port, ships2 as ship, parameters as p
24  where earliest_arrival(ship.longitude, ship.latitude,
25            port.longitude, port.latitude, ship.maxspeed) <= p.deadline;
```

The syntax used in the script is that of PostgreSQL. In this example, the script includes a user defined function (i.e. `earliest_arrival`) that computes the time for a ship to reach a port given their coordinates and the ship's speed. Each task can be associated with any number of user-defined functions and at least one *select-into* statement that would store the outcome of the computation on a (temporary) table, to be consistent with the intent specified in the conceptual model. In Listing 3.1, the *select-into* statement defined in lines 11-17 takes as input tables `ports2`, `ships2` and `parameters` (highlighted in line 15) and stores its result in table `reachable_ports2` (line 14). Similarly, the *select-into* statement defined in lines 19-25 takes the same input tables and stores its result in table `ship_requirements2` (line 22). Both statements, in turn, call the function `earliest_arrival`, which is defined in lines 1-9.

SQL collaboration workflows are composed of multiple SQL workflows, each associated with one participant pool. Each one of those SQL workflows may include sophisticated constructs to captures conditional branching and concurrency, as per the BPMN standard. The disclosure analysis is performed over the whole SQL collaboration workflow. To that end, we map a SQL collaboration workflow into a Petri net for analysis purposes. Petri nets is a well-known formalism for modeling and analysis of concurrent systems.

The Leaks-When analysis of a SQL collaboration workflow proceeds in several stages as sketched in Fig. 16. Assuming that a SQL collaboration workflow is provided, the first stage consists in translating the BPMN collaboration into a Petri net. In the second stage, an unfolding of the Petri net is computed which explicitly represents all the possible executions of the Petri net, a.k.a. runs. This unfolding is acyclic, but it explicitly captures one iteration of every loop in the original model. Then, the technique concatenates all the SQL statements attached to each one of the nodes in the SQL collaboration workflow to generate a SQL script for each run. Finally, the technique generates a leaks-when report for each output data object in the SQL collaboration workflow as requested by the analyst. Below we describe each of the above stages in turn.

3.2.2.2 Mapping SQL Collaboration Workflows to Petri Nets.

The disclosure analysis takes as input models specified as BPMN collaborations that we need to translated into Petri nets. Fortunately, the mapping of BPMN process models into Petri nets has been described in [29]. For completeness, we include the transformation rules in Fig. 17.

Note that the transformation rules cover only the mapping of BPMN process models (SQL workflows). However, the mapping can be straightforwardly adapted to our context as follows. We first map the SQL workflow from each one of the pools in the collaboration model. Then, we add a place for each message exchange in the collaboration. Such a place will be connected with an arc with a source on the transition that represents the message sending task in one of the SQL workflows and with a target in

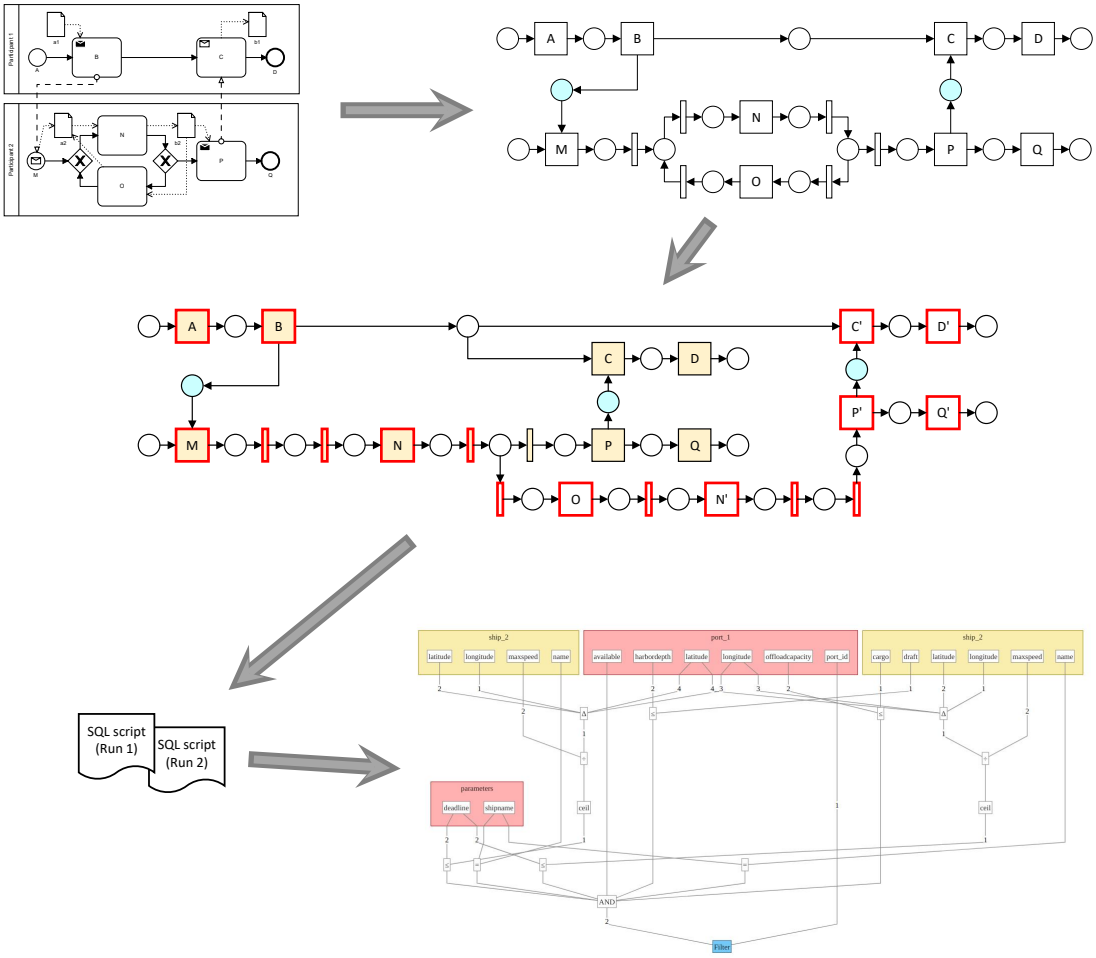


Figure 16: Stages in the Disclosure Analysis

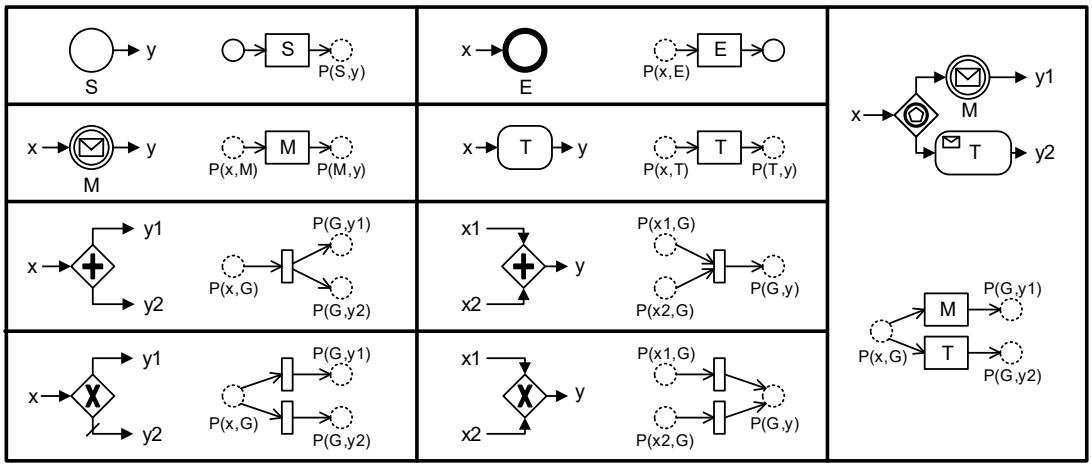


Figure 17: BPMN to Petri Nets: Transformation Rules

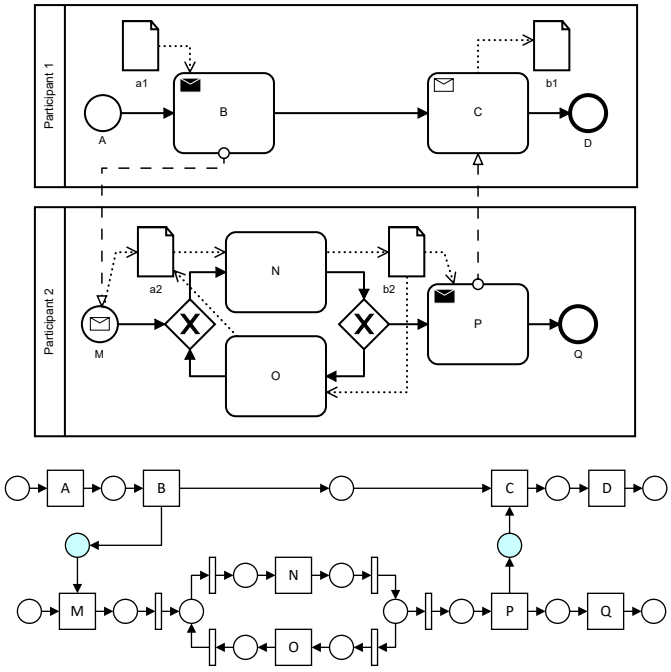


Figure 18: An example of the Mapping of a SQL Collaboration Workflow to Petri Nets

the place. Similarly, the place will be have an arc that stems from itself and targets the receiving task in another pool. Fig. 18 shows side-by-side a sample SQL collaboration workflow and its corresponding Petri net. The places representing message exchange are shown with a filling of color blue to help one in their identification.

The transformation keeps the information of the mapping such that, given a task in the SQL collaboration workflow it is possible to determine the transition in the Petri net and vice versa. It is worth noting that the transformation covers only the control flow perspective of the collaboration. However, by using the bidirectional mapping of BPMN tasks into Petri net transitions, it is possible to determine the input data objects (e.g. "a1" in the case of task "B") and also the output data objects (e.g. "b1" in the case of task "C") whenever required.

3.2.2.3 Computing the Set of Runs of a SQL Collaboration Workflow. In [30], we presented a technique that can be used to compute the set of runs on SQL workflows without loops. We argued that the extension of the technique to the case where the SQL workflow include loops can be achieved by implementing and strategy for unrolling the loops to explicitly represent one iteration. Nonetheless, the potential presence of concurrency and mutual exclusion in the SQL workflows requires a specialized machinery. In fact, extending the technique from single SQL workflow to a collaboration, introduces per se some difficulties, since execution of the SQL workflow on each one of the pools can evolve independently, i.e. a SQL collaboration workflow describes naturally a system with concurrency. That is the reason why here we chose to map the SQL collaboration workflow into a Petri net.

Once with a Petri net, we leverage existing work to compute the set of runs in the system. To that end, we first apply the technique described in [31] to compute the unfolding of the Petri net. An unfolding of the Petri net is another net that explicitly represents the computation specified by an input Petri net. More specifically, an unfolding will incrementally add transitions and places to represent the execution of the original Petri net. Henceforth, an unfolding of a Petri net is potentially infinite if the input net has loops. However, techniques such as the one presented in [32] have been devised to truncate the unfolding in a point where the behavior of the original Petri net is explicitly represented by a prefix of the unfolding. In fact, the prefix introduced in [31], referred to as the complete prefix unfolding, is suitable for tasks such as model checking. However, the complete prefix unfolding does not represent

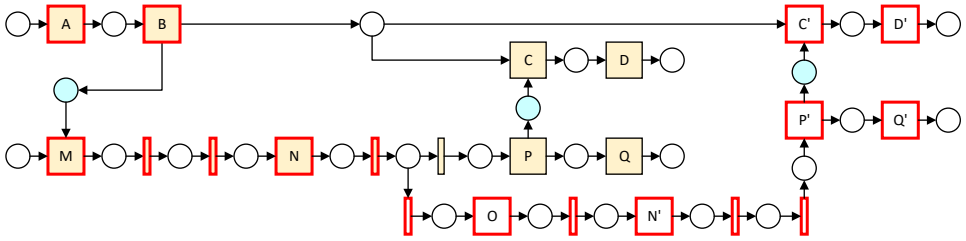


Figure 19: Runs of the SQL Collaboration Workflow from Fig. 18, Represented as the Unfolding of its Petri Net

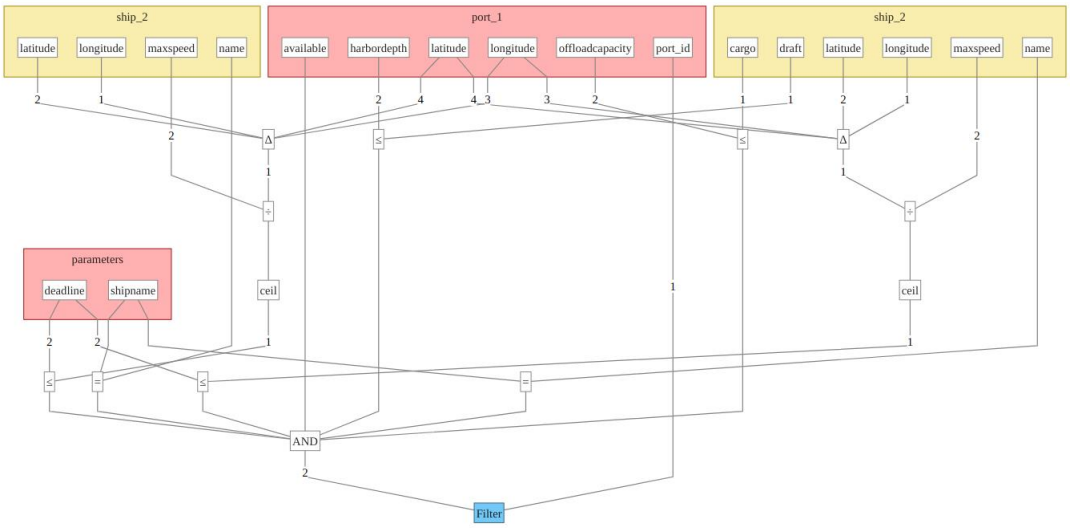


Figure 20: Sample Leaks-When Report

loops explicitly in the way we need. It is for this reason that we use the truncation criterion defined in [32]. The unfolding generated using such truncation criterion for our running example is shown in Fig. 19.

From the unfolding in Fig. 19, we can see that the SQL collaboration workflow has two runs. To help in identifying the tasks involved in each one of the runs, we have colored them: the first run includes all the tasks (transitions) filled in yellow, the second one includes all the tasks with borders in red. As hinted by the labels in the unfolding, it is also possible to track back the transitions to the corresponding tasks in the original SQL collaboration workflow. Indeed, we can keep such mapping in a data structure as required.

For each one of the runs, we can straightforwardly derive a script by concatenating all the SQL statements of the tasks and data objects in the SQL collaboration workflow. Since some of the tasks in a run can be concurrent, a topological order can be used for deciding an order in the SQL statements added to the SQL script. Each one of this scripts is then passed to the SQL workflow analysis presented in Sec. 3.2.1.

The result of applying the SQL workflow analysis to the script given in Listing 3.1, considering the release of `port.port_id`, is given in Fig. 20. It shows what is leaked (`port.port_id`), and when it is leaked. The condition is a conjunction of several sub-conditions, related to the existence of ships with a given name that fit into this port and can reach it before the given deadline.

3.2.3 Privacy Policies in SQL Collaboration Workflows. Different participants in a SQL collaboration workflow may have different privacy policies on data they own. We have thus extended our

workflow analysis in order to specify and take into account such policies. Having such policies, it is possible to speak about and report any eventual disclosure of sensitive data that is unexpected and/or undesired. During the Leaks-When analysis, we run the policy compliance checking and highlight the potential disclosures in the Leaks-When report with respect to selected party.

Consider the same Aid Distribution scenario in Fig. 15. The scenario implies an incremental disclosure of the sensitive information, such as coordinates of the marine ships, which are transporting the goods provided by international community. The goal is to disclose certain data only when it becomes allowed by the collaboration process constraints.

In the model Aid requesting country first shares the available ports and the deadline for the goods delivery. Then Aid providing country checks what ships can reach the available ports by the given deadline and shares the requirements for available ships to be able to allocate it to the ports. These requirements are cargo and length of the ship. After that Aid requesting country selects feasible ports, depending on the ports' harbor depth and offload capacity, it sends this filtered list to Aid providing country and then Aid providing country eventually can select the most fit ship for the most feasible port (e.g. the closest) with right dimensions to allocate it to the free berth of the selected port. In the final phase all data about the selected ship is disclosed to the Aid requesting country, including ship name, coordinates, id, cargo, draft and maximum speed. All aforementioned disclosures are supposed to be made via BPMN Message Flow.

3.2.3.1 Privacy Policies. Without privacy policies, the user / analyst has to manually discover the potential process violations coming from the permissions of the parties, on the basis of the leaks-when reports similar to the one in Fig. 20. If anything in the SQL workflow has been disclosed from one BPMN Pool to another by BPMN Message Flow, user has to figure it out by him/herself. To hand over more control and automation to the analyzer, we support the *privacy policy extension*.

Privacy Policy notation and binding. We implemented an extension to the SQL policy specification, so the privacy policy is represented by the SQL `grant` statement and serves as an established constraint for the data sharing during the process execution. In the collaboration SQL workflows we assume that the concept of a party corresponds to the BPMN Pool. The parties to be used in the SQL `grant` statement are extracted from the BPMN Pools' names of the business process.

Since all of the disclosures in the SQL workflows are currently occurring through the `select into` statements, the policy gives a permission for the selected party to run `select into` queries over the specific attributes of the SQL table or the entire table. Such approach is equal to the 'read' access right. By default we assume that all of the SQL tables are prohibited for the read access.

As a starting point of building up the policy for specific party we can use the context menu of the BPMN Pool and attach an SQL script. In the Aid Distribution scenario we acknowledge that the 'parameters' table is fully accessible from the beginning of the process execution and to the end event. We call such policy rules 'Global' and attach them to the BPMN Pools. Fig. 21 demonstrates such a *Global* privacy policy rule.

Since the names of the BPMN Pool may appear too diverse and contain the undesired symbols, we assume that the inner representation of the party name is a subset of literal words in a lowercase joined with an underscore. For instance, the '1. Aid requesting country party' must be mentioned as an 'aid_requesting_country' in the policy rule.

In the Aid Distribution scenario it is not secure to disclose all of the sensitive information at once, that is why an incremental approach has been chosen. This approach also implies that during the process execution the permissions of the party may extend. Hence, additional policy rules can be attached dynamically. To achieve this the process creator can select an output data object of a task and assign an SQL script with the `grant` statement of a policy rule. By doing this, we infer that to produce the selected output without violations a party is endowed with this additional policy rule which will hold until the process end event. We call such a policy rule 'Local' and assign it to the output data objects of the BPMN Tasks. Fig. 22 demonstrates such a *Local* policy rule.

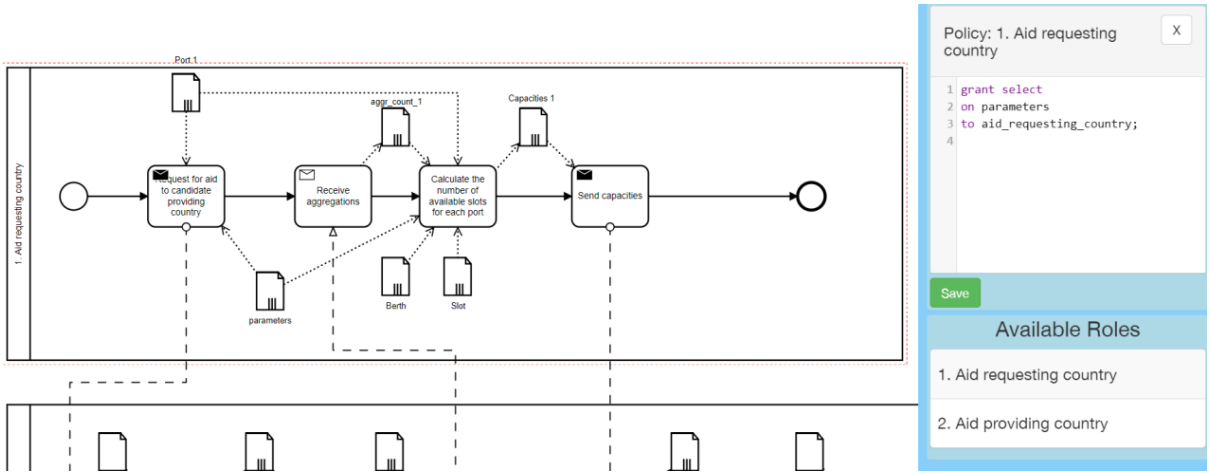


Figure 21: *Global* Policy Rule

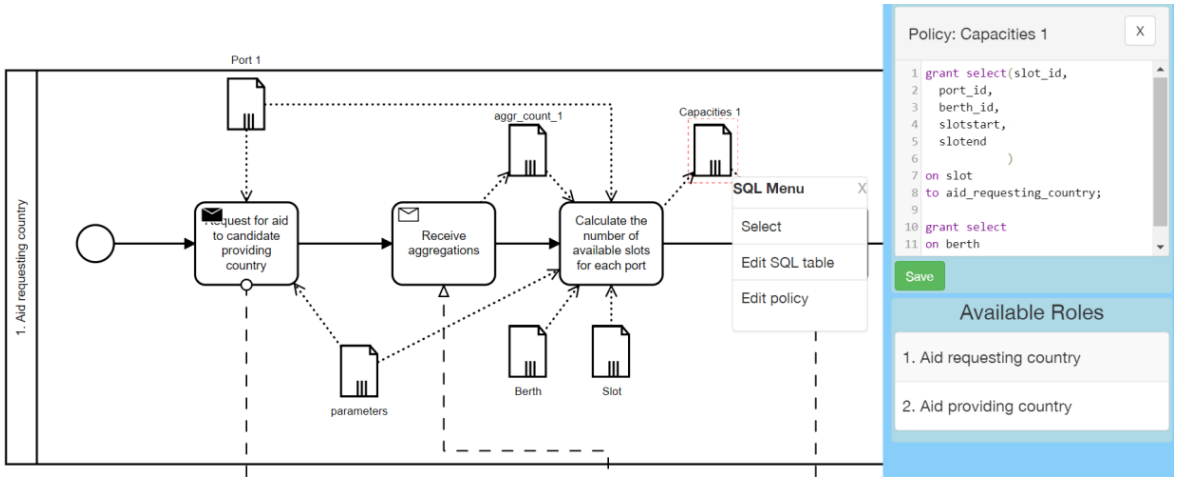


Figure 22: *Local* Policy Rule

Hence, in addition to the incremental disclosure of the sensitive attributes we implemented incremental extension of the policy ruleset for the party during the process execution. Such rules are allocated after some point of the process passed and hold until the process end.

Privacy Policy limitations. In order to be parsed and applied correctly, the privacy policy must use a SQL 'grant' script in specific manner demonstrated in the listing 3.2.

Listing 3.2: Example of the Privacy Policy Rule Using SQL "grant" Statement

```
grant select(ship_id,
             latitude,
             longitude
)
on ship_2
to aid_requesting_country;
```

In case of the specification of concrete SQL attributes the first attribute must be placed on the same line with the 'select' keyword following other attributes each on the new line. The syntax used in the script is that of PostgreSQL. On the Fig. 23 we provide a formal representation of the privacy policy syntax in the Backus-Naur form.

Modifications to the Leaks-When Analysis. In order to make the Leaks-When analysis work with policies, we have adapted it as follows:

$$
\begin{array}{rcl}
\textit{privacyPolicy} & ::= & \textit{privacyPolicyRule EOL} \\
& | & \textit{privacyPolicyRule EOL privacyPolicy} \\
\textit{privacyPolicyRule} & ::= & \textit{grant select EOL on } \{\textit{tableName}\} \textit{ EOL to } \{\textit{partyName}\}; \textit{ EOL} \\
& | & \textit{grant select}(\{\textit{sqlAttributeName}\} \textit{ sqlAttributesListTail EOL})\textit{EOL} \\
& & \textit{on } \{\textit{tableName}\} \textit{ EOL to } \{\textit{partyName}\}; \textit{ EOL} \\
& | & \text{””} \\
\textit{sqlAttributesListTail} & ::= & , \textit{ EOL } \{\textit{sqlAttributeName}\} \textit{ sqlAttributesListTail} \\
& | & \text{””}
\end{array}
$$

Figure 23: Privacy Policy Syntax

1. The Leaks-When analysis is executed with respect to the party-holder of the selected output data object.
2. The policy rules are aggregated from the BPMN Pool of the party-holder up to the selected data object. To have the right ordering of the privacy policy rules we employ the Petri net generated for the business process using techniques described in Sec. 3.2.2.2.
3. The Leaks-When report highlights potential disclosures with colors.

The Leaks-When analyzer uses the specification of the privacy policy rules to carry out the verification over the SQL Collaboration workflow. If several data objects are selected, then the analyzer will still execute role-wise analysis for each of the data objects. In the Leaks-When report the edges are colored with a red color in case of 'Direct' disclosure, when the attribute is referred explicitly in the 'select' statement. Orange color is used for 'Indirect' disclosure, when attribute is used in the calculations, over the SQL aggregation functions or as a parameter of a user-defined function. In such case the attribute still can be inferred employing differential privacy technique.

Fig. 24 demonstrates the Leaks-When report with the privacy policy highlights.

As we can see, all of the successors of the operation that might disclose the attribute are marked orange up to the root 'filter' operation. The attributes 'latitude', 'longitude' and 'maxspeed' from the SQL table 'ship_2' are not directly used in the final select (a.k.a. Filter in the Leaks-When report). The they are used in the calculation of the distance between the 'port_1' and 'ship_2', so there is a relation which can be employed by the adversary.

3.2.3.2 Aid Distribution Scenario Extension with SQL Aggregation Functions. In the Aid Distribution scenario during the first step of the workflow called 'Compute reachable ports' we select a subset of the ships that can reach the requested port on time according to the 'parameters' SQL table. We send the information about their arrival times and cargoes to the Aid requesting country. However, if the port capacities are not big enough, some of the ships can be rejected because port cannot allocate so many of the proposed ships. Hence most of the disclosed information about ships is excessive.

For instance, Aid providing country possesses 15 ships that are ready and sufficiently close to the given port. However, the port can match as much as 5 slots available for allocation, while Aid providing country is unaware about this limitation and still discloses arrival times of all of the 15 ships. These arrival times are narrowing down the area where the ship can be present, especially if adversary party tries to match arrival times of the ships with their cargos. In case of only one port adversary can infer only the radius of the ship location, but using multiple queries with different ports it is possible to carry out the triangulation and guess the location more precisely.

We can diminish this issue by inserting another interaction step in the beginning of the process. Fig. 25 demonstrates this extension to the main model.

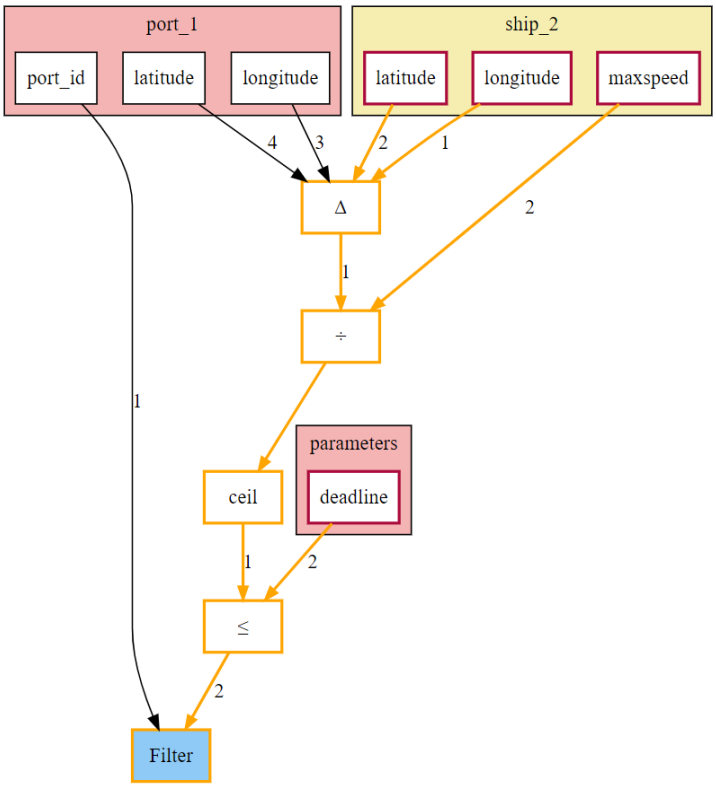


Figure 24: Leaks-When Report with Applied Privacy Policy

The aforementioned process model is also different from the main Aid distribution model because of presence of the SQL aggregation functions. The tasks 'Count the number of reachable ships for each port' and 'Count the sum cargo of reachable ships for each port' are using aggregation functions and does not disclose any single ship arrival date or cargo. After this step the Aid requesting country receives only aggregated count of the ships and their aggregated cargo for each port. Henceforth, Aid requesting country can match them with own port capabilities. After the task 'Count the number of available slots for each port' Aid providing country receives the capabilities of the ports and can now narrow down the list of the ships required to satisfy maximum of the the port slots. On the other hand, the information about currently available slots is also sensitive data, so if the number of available ships is smaller than the number of slots, the potential adversary party will know only about the matched number of slots.

Listing 3.3 presents the SQL script assigned to the task 'Count the number of reachable ships for each port'.

Listing 3.3: Task From Step 1

```
1  create or replace function aggr_count(portname TEXT)
2    returns TABLE(cnt INT8) as
3  $$
4    select count(ship_2.ship_id) as cnt
5    from ship_2, port_2, parameters
6    where port_2.name = parameters.portname
7          AND (point(ship_2.latitude, ship_2.longitude) <@>
8               point(port_2.latitude, port_2.longitude))
9          / ship_2.max_speed <= parameters.deadline
10 $$
11 language SQL IMMUTABLE returns NULL on NULL INPUT;
12
13 select p.name as name, res.cnt as cnt
14 into aggr_count_2
15 from port_2 as p cross join aggr_count(p.name) as res;
```

In this example, the script includes a user function `aggr_count` defined in lines 1-9 that computes

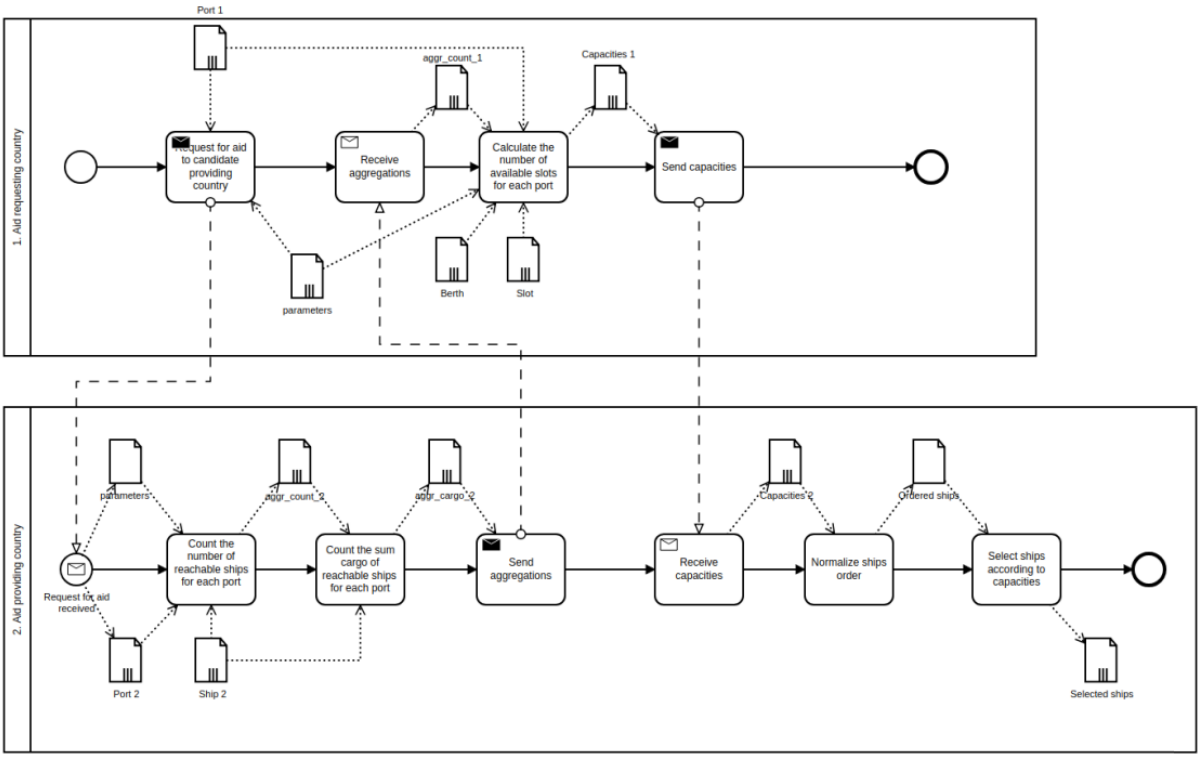


Figure 25: Extension for Aid Distribution Scenario

the number of ships that can reach a given port by the given deadline having their coordinates and maximum speed. Since every task must be associated with at least one *select-into* statement the task writes the outcome of the computation to the temporary table `aggr_count_2` in the lines 11-13.

Currently supported group operations are following:

1. count - returns a number of rows with a non-empty given attribute; count(*) computes a number of rows with no respect to specific attribute, in current implementation takes an 'id' attribute of the SQL table.
2. sum - returns a sum of a given attribute of all rows.
3. avg - returns an average value of a given attribute of all rows.
4. max - returns a maximum value of a given attribute of all rows.
5. min - returns a minimum value of a given attribute of all rows.

Fig. 26 demonstrates the Leaks-When report for SQL count function.

On this graph we select a subset of the records from the 'port_1' SQL table, which match the given 'portname' from the parameters. Then we match each of the selected ports with all of the rows from the 'ship_2' table and calculate the distance between the port and the ship. We divide each distance by the respective ship 'max_speed' and filter out those pairs, whose travel time exceeds the given deadline from the 'parameters' SQL table. The 'Filter' operation for the 'ship_id' attribute which precedes the final 'count' aggregation function is a special way of handling 'count(*)'. That is why it is marked with red color - it started to be used internally, without user's intention and policy rule. The operations related to the 'latitude', 'longitude' and 'max_speed' attributes are marked with orange color, because they are not directly disclosed. But they still can be guessed using multiple queries with the different ports input, which stands for employing the differential privacy algorithms.

3.2.4 Analysis of BPMN. We have used BPMN to specify collaborative SQL workflows, and then apply the leaks-when analysis to them. We would also like to analyze the business processes "directly",

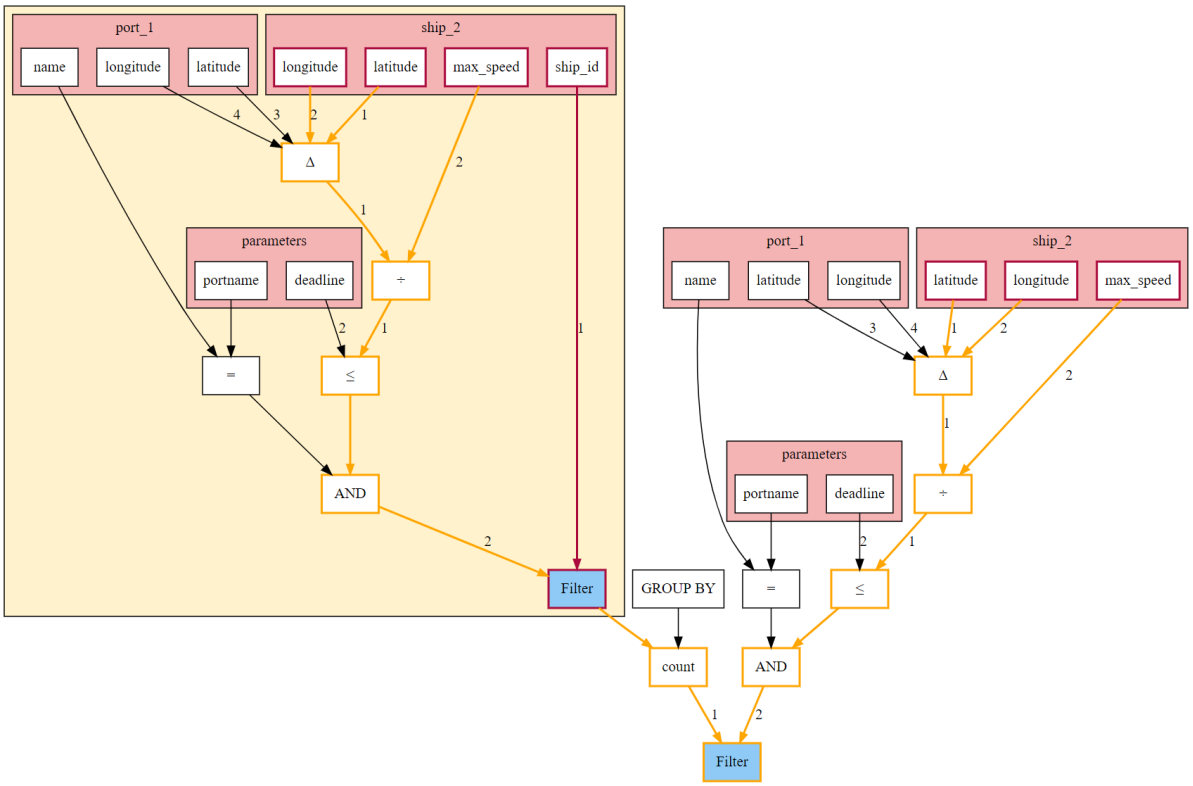


Figure 26: Leaks-When Report for Aggregation Function with Privacy Policy Applied

considering each task as an operation on the data objects, from and to which there is data flow, considering the control flow and events the process may include, encoding the processes directly into SDGs, and doing the leaks-when analysis on them. Below, we will describe this encoding, the analysis, and the presentation of analysis results. We will see that the understandability of the analysis result very much depends on how much the imperative, "control-oriented" features of BPMN have been used.

Once the to-be-analyzed computation has been converted to a SDG, the task of the analyzer is to simplify it as much as possible. For SQL workflows, we have displayed the result of the simplification rather directly to the user of the PLEAK tool, allowing the user to visually determine the actual dependencies. With large SDGs, such display no longer serves a purpose. We have thus chosen to report to the user, which of the inputs may flow to which of the outputs, and through which identified intermediate nodes (that may represent some filterings or sanitations) it must pass.

3.2.4.1 Business Processes, Their Active Components, and Analysis. To explain what we mean by the leaks-when analysis of a business process, let us consider the example process in Fig. 27. It is more or less one of the processes that model the possible activities in RapidGather, developed in the Mobile CRT of Brandeis.

The process has a number of tasks that read and write a number of datasets. Each dataset has one or more *data fields*. In our example, we are going for simplicity and hence let each dataset to have just a single field `v`. Each task that has a number of datasets as its inputs and outputs, computes the fields of output datasets from the fields of input datasets, and nothing else. Each task that precedes an exclusive gateway (in Fig. 27, the task labeled "Check for permission" in the "Mobile device" pool) computes a boolean value that is used by the following gateway. Each task is annotated with the code that does the computation, e.g. the specification of the code for task `T71` is depicted in Fig. 28. Note that several tasks could in principle write into the same dataset (although that is not the case in our example here).

In principle, the programming language in tasks could be quite complex, e.g. SQL, but currently we have settled for a simple language that defines the fields of output datasets as expressions computed

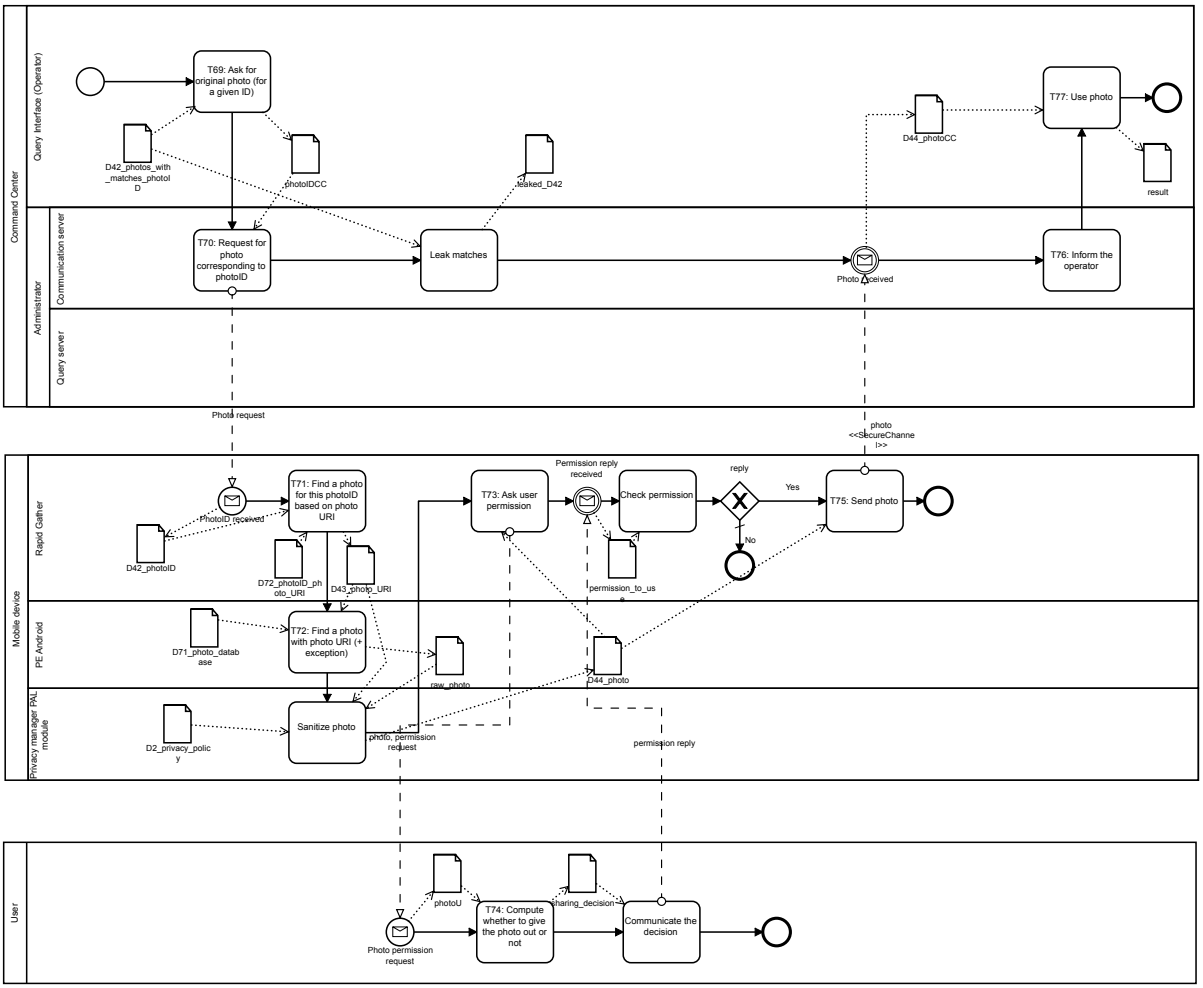


Figure 27: Example Process for Leaks-When Analysis of BPMN

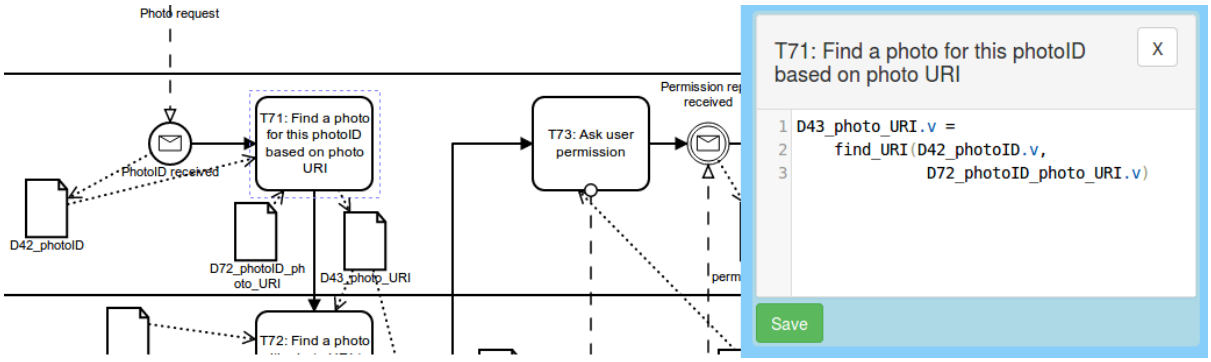


Figure 28: The Code of a Task

s	process name
p	pointer to subprocess
d	dataset name
op	arithmetic, logical, etc. operation

R	running process
P	inactive process
L	process launcher
T	task
E	arithmetic, logical, etc. expression

$$R ::= \mathbf{0} \mid T \mid R_1 \,;\, R_2 \mid R_1 \parallel R_2 \mid E \,?\, R : R \mid (P)^*_s \mid L$$

$$P ::= p.d_1, \ldots, d_k : R$$

$$L ::= p \rightarrow \uparrow\uparrow \cdots \uparrow s_1 s_2 \cdots s_k$$

$$T ::= d_1 \leftarrow E_1; \ldots; d_k \leftarrow E_k$$

$$E ::= d \mid p.d \mid \mathbf{op}(E_1, \ldots, E_k)$$

Figure 29: Intermediate Calculus for Translating From BPMN to SDGs

from input datasets, with arbitrary names for the functions.

Fig. 27 contains three processes talking to each other. We consider each of them to be replicated, i.e. they can be started many times. With many processes running, we want their message exchanges to be organized. E.g. the message sent by the task "Communicate the decision" from the user should go back to the same mobile device process that started the user process. In Sec. 3.2.4.5 we describe how we recover this correspondence.

The leaks-when analysis also requires us to specify, what are the inputs and what are the outputs of the process. Currently we are picking as inputs the datasets that are never written to, and as outputs the datasets that are never read from.

3.2.4.2 An Intermediate Process Calculus.

Syntax. To make the translation from BPMN to SDGs conceptually simpler, we split it into two steps. We thus define an intermediate representation, which still retains the imperative features of the BPMN, but also does away with some other features, in particular the message flows between the pools. The syntax of the intermediate process calculus is given in Fig. 29.

Here a running process R is either a process that has finished its work, a task T that does computations, or one process followed by another one, or two processes running in parallel, or a choice between running two processes, depending on the outcome of the expression E. It can also be the making available of a pool of processes P, known under the name s, which can be subsequently launched by either the process that follows this action, or by other processes running in parallel. Finally, a running process may launch other processes. We note that our process calculus has retained the synchronization points of BPMN — in a running process $(R_1|R_2); R_3$, both R_1 and R_2 have to finish before R_3 can start running.

An inactive process P consists of its body R, as well as the datasets $d_1, \ldots, d_k$ that are local to this processes. When the process P is launched, then the process pointer p will point to the process that launched it. This process pointer can be used to access the datasets of the launching process. A process launcher L is a pair of the address of the to-be-launched process, and a new process pointer p. After executing L, the pointer p points to the newly launched process. The address consists of a path through the process tree, first going up, and then going down by following the names of the process pools.

The tasks update the values of the datasets d. We call them datasets, because they will correspond to the datasets of BPMN, but in our calculus they are rather treated as variables, where the internal structure of the values they contain is not relevant for the calculus. If the internal structure is important, then the operations **op** have to reflect that, and the simplifying transformations of the analyzer have to know about the identities these operations satisfy. A task can refer either to the datasets local to the

current running process, or to datasets of other processes for which it has process pointers. Only local datasets can be written to.

We see that the process pointers p are the mechanism for different subprocesses to communicate. In this sense, they are similar to the channels used in various process calculi, e.g. π-calculus [33] or join-calculus [34]. The use of pointers in our calculus is more restricted; this helps with the translation to SDGs, as we can statically identify, which processes a pointer may point to. In the actual communication of values between processes, the reader takes the active part. Again, this simplifies the translation into a SDG. We have had good experience with similar constructions [35].

About the Semantics. Let us give a sketch (but not a full formalization) of the small-step operational semantics of the calculus. The main notion of an operational semantics is that of the *configuration*, representing the current state of the execution. Also, the semantics defines a relation on the set of configurations, expressing the possible ways of a configuration making a single step and becoming the next configuration.

A configuration is a tree of nodes. Each node contains a running process, as well as the pools of activatable processes, the values of its datasets and the process pointers it has defined. Each process pointer points to some other node in the tree. Initially, the tree has a single node, containing the topmost process and the datasets that are local to it. If some node N contains a pool of activatable processes $(p.d_1, \ldots, d_k : R)^*_s$, and some other node N' wants to activate a process from this pool, then a new child node N'' of N is created, containing the running process R and the datasets $d_1, \ldots, d_k$, with some default, initial values. The node N'' also contains the pointer p which points to N'. The arc from N to N'' is labeled with s. On the other hand, the process in node N', which has just executed a process launcher, now has a process pointer pointing to N''.

A process in a node N can access the datasets in node N, as well as all datasets in its ancestor nodes. Such access includes the ability to write into these datasets. Similarly, if the datasets in node N are read through a pointer pointing to N, then one can also read all datasets in the ancestor nodes of N through the same pointer.

If a process is about to execute a task, then it can always do it (as long as the process is syntactically correct). It can access the datasets via the pointers it has, and can write to local datasets. The execution of a process launcher $\uparrow \cdots \uparrow s_1 \cdots s_k$ in a node N' works as follows. We first find the node N' that has a pool of activatable processes $(P)^*_{s_k}$. To find it, we walk upwards from the node N as many steps as there are arrows in the process launcher. We will then walk downwards along the arcs labeled $s_1, \ldots, s_{k-1}$, reaching a node N. It is possible that some node has several outgoing arcs labeled with s_i; in this case, we non-deterministically select one of them. If the node N contains a pool of processes with the name s_k, then a new node N'' is created and the pointers are defined, as described above.

3.2.4.3 Translating the Process Calculus to SDGs. Compared to the conversion of SQL workflows to SDGs, the main difficulty of translating from the calculus in Fig. 29 to SDGs is dealing with the datasets that are written to in several places of the calculus. Another, independent complication is the translation of the process starts and process pointers.

We will not give a full description on how a process is translated to a SDG. But we will describe the main issues here.

3.2.4.4 Hierarchy of Processes. In the process calculus, each process may have replicated subprocesses. All the possible instances of these processes have an address, consisting of an identifier for this instance under the immediately containing process, as well as the full address of the containing process. These addresses immediately map to the coordinates in a dependency graph, and the names of the sets of processes become the names of the dimensions in the SDG. Given a full running process R, we can say for each of its subprocesses and tasks, which set of dimensions they have.

Execution and Timing. In a process, some parts may get executed and others may not. Also, the order in which the parts are executed, matters. Hence, whenever we translate a process into SDG, the resulting graph has two special nodes. These nodes have the same dimensions as the converted process. One of them returns a boolean value, indicating whether this process instance ran till the end. The other returns a timepoint (we model the timepoints as integers), indicating the time when this process finished. The value of the first is computed from the results of the guards in the converted process, while the value of the second one comes from the order of reads and writes of persistent datasets.

Pointers to Subprocesses. Each process launcher L inside a subprocess P with certain dimensions D launches a subprocess P' with different dimensions D'. We want to state that different instances of L (they have different coordinates) launch different instances of P'. We thus define mappings $\mathbf{Addr}_L$ from the coordinates of P to the coordinates of P'. At the start of the process P', the values from $\mathbf{Addr}_L$ for all possible coordinate values of L are checked (and also checked for any other process launchers L' that may launch P') and the execution and timing nodes for the start of P' are set up.

The mappings $\mathbf{Addr}_L$ are treated as uninterpreted functions by our analysis. However, the analysis takes into account that they are injective, and could thus simplify e.g. some conjunctions of equality checks.

The mappings $\mathbf{Addr}_L$ are used when converting the access of a dataset through a process pointer. If the pointer p, coming from the launcher L, and inside a process with dimensions D is used to access a dataset d' in process P' with dimensions D', then we compare $\mathbf{Addr}_L(\mathbf{c})$ against $\mathbf{c}'$ for all possible values of $\mathbf{c}'$, where $\mathbf{c}$ is the tuple of coordinates of the current process instance, and $\mathbf{c}'$ ranges over possible tuples of coordinates in dimensions D'. We use the results to *filter* the dataset d'. A filtering operation takes two inputs, the second of them boolean, and returns the first input if the second input was true, and $\perp$ otherwise. Note that the filtering results have the dimensions $D \| D'$, but for each value of the coordinates of D, at most one of the filtering results is not $\perp$. We will then use a *merge*-operation [27] to select this non-$\perp$ result.

Persistent Datasets. We call a dataset *persistent* if it is written into by at least two different tasks. To record the timepoints when a read or write is done, we again define for each reading or writing task T an uninterpreted function $\mathbf{Moment}_T$. Again, this function takes a tuple of coordinates as its input, returns a timepoint, and is injective. But to better argue about the ordering of reads and writes, as long as this ordering can be derived from the structure of the process, the functions $\mathbf{Moment}_T$ also take a second argument — a timepoint. The output of the function is larger than the value of this second argument. When converting the process, the second argument will be the maximum of the timepoints of preceding reads and writes to persistent datasets.

When converting a writing to a persistent dataset, the SDG will just contain a node v_w with the $\mathbf{Moment}_T$-function. Also, the conversion will remember the node v_g that generated the value that was written. When reading a persistent dataset, the SDG will contain a node v_r with the $\mathbf{Moment}_T$ function. The value of the v_r node is compared against the values of all v_w-nodes. The results of the comparisons are used to filter the values from the v_g-nodes. A merge-like operation is then used to select the value with the largest timestamp from among the v_g-nodes that were not filtered out.

3.2.4.5 Translating from BPMN to the Process Calculus. The conversion from actual process expressed in BPMN consists of first rewriting the process, removing from it a number of BPMN constructions. These constructors mostly express exceptional behavior, or structure the conceptual steps of the process. The second step of the conversion rewrites the message flows. A message flow that starts a subprocess, becomes a process launcher. Also, at the receiving side of each message flow, we introduce the the task copying the sent dataset to a local dataset.

For the first step, we have introduced the following transformations:

Single end events for subprocesses. For subprocesses with end events of multiple types (normal and exceptional), introduce a dataset into which the type of the reached end event is written. This

dataset can be read by the tasks following the subprocess. In this way, instead of one or more boundary events of the subprocess, starting the processes that handle the exceptional outcomes, there will be a check of the value of the introduced dataset, followed by an exclusive gateway.

Conditional gateways to exclusive ones. When the control reaches a conditional gateway, it stops and waits for one of the events in the following branches to occur. The execution then continues along the branch with the event that happened first. In the transformation, we introduce a dataset that records which event happened first. The branches with the events write into this dataset and then stop. The actual continuation of the process is moved to a separate branch, that starts with the check of the value of this dataset, and then branches according to it.

Dissolve non-replicated subprocesses. This change is straightforward to make. We redirect the sequence flows going into the subprocess, making them go into the start event instead. Similarly, the sequence flows from the subprocess are made to start from its end event instead. The plain start and end events are then removed.

Tail-call optimization. If the last step of one subprocess is to send a message that invokes a new instance of a different subprocess, then we merge these two processes.

In the second step, we have to trace how the message flows cross the hierarchy of subprocesses. Based on this distance between the sending task and receiving event, and comparing it to the distances of the message flows that started either this or other subprocesses, we decide which message flows belong together. Such groups of message flows that belong together should have the same process pointers. Then we perform the transformations described in the first paragraph of this section. When grouping the message flows, we are actually creating channels, which the BPMN notation does not originally have.

3.2.4.6 New Transformations in SDGs. The introduction of the *merge*-nodes requires us to also introduce simplifications of SDGs that either remove these nodes or move them to a place in SDGs where they are not in the way of us deducing the validity of the facts that we wanted the leaks-when analysis to verify. The simplifications allow the *merge*-nodes to be moved over the edges of the SDG, taking into account that they behave as identities with certain further complexities. These transformations have been described in [27].

3.2.4.7 Analysis of the Example Business Process. When converting the process in Fig. 27 into a SDG, and simplifying it, we end up with the graph in Fig. 30. This figure is still pretty readable. We see that the input `photo_database` may leak to the output `result`, but only after passing through the operation `filter_sanitize_photo`. We can also deduce that in order for this output to be active, the check `is_photo_innocent` must pass.

In general, we expect the resulting dependency graphs to be much more complex. Hence we deliver the results of the leaks-when analysis in the form of stating for each input and each output, whether this input may flow into this output. Moreover, if the flow is possible, then we state through which operations it must flow, and which checks must pass. The list of these significant operations and checks will be another input to the leaks-when analyzer.

3.3 Quantitative Analysis

A *quantitative privacy analysis* allows us to numerically characterize the privacy losses in a process. Being more complex than the analyses described above, we generally apply them to simpler workflows. The expressivity of the workflows that we mostly consider is similar to the workflows to which the BPMN leaks-when analysis was applied (see Sec. 3.2.4.1). Such workflows take data objects as inputs, and compute new data objects from them. The workflows may be composed either sequentially or in parallel, where we combine the control structures of the two workflows, as well as let certain input and output datasets of the first workflow to be certain outputs or inputs of the second. A workflow is made

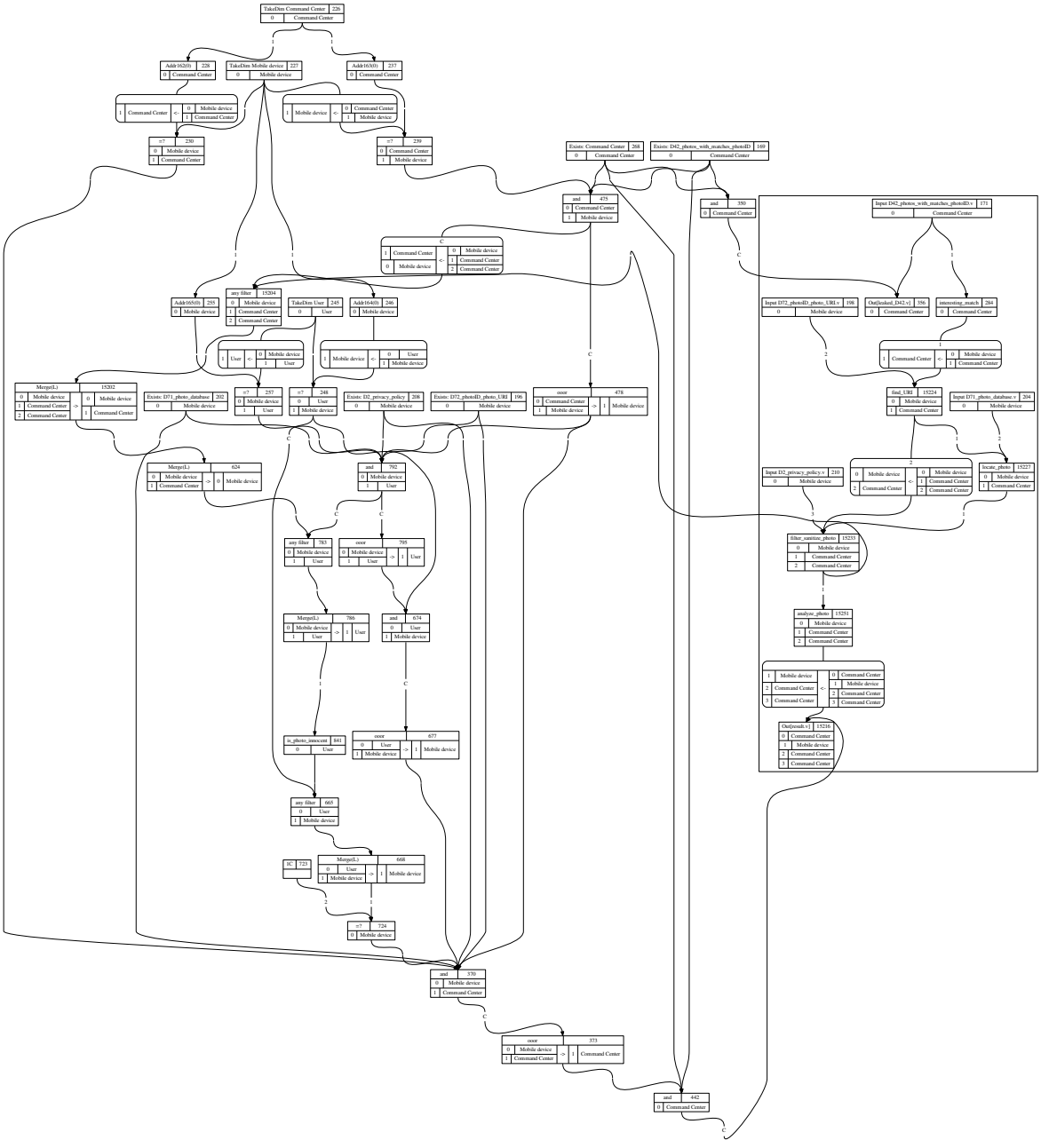


Figure 30: Final Dependency Graph for Our Example

up of components; we want to analyse both the components, as well as the workflow as the composition of components.

Below, we present a large number of different analyses, applied to slightly different formulations of workflows, and to different definitions of privacy leaks. For each analysis, we define the precise shape of the workflows it can be applied on. It is generally possible to convert between different kinds of workflows we consider, although it may incur some loss in precision or expressivity.

3.3.1 Sensitivity and Differential Privacy.

3.3.1.1 Notation. We let $\mathbb{N}$ denote the set of natural numbers, $\mathbb{R}$ the set of real numbers and $\mathbb{B} = \{\mathsf{true}, \mathsf{false}\}$ the set of booleans. Let $\mathbb{R}_+$ denote the set of non-negative real numbers, augmented with the greatest element ∞. For $a, b \in \mathbb{R}$, $a \leq b$, let $[a, b]$ denote the set of real numbers x, such that $a \leq x \leq b$.

For values x and y, the *Kronecker symbol* δ_{xy} is equal to 1 if $x = y$, and to 0 otherwise.

The set of all subsets of the set X is denoted by $\mathcal{P}(X)$. If X is a partially ordered set and $X' \subseteq X$, then the *upper closure* and the *lower closure* of X' are defined by $\uparrow X' = \{x \in X \,|\, \exists x' \in X' : x \geq x'\}$ and $\downarrow X' = \{x \in X \,|\, \exists x' \in X' : x \leq x'\}$. A subset $X' \subseteq X$ is *upwards* [resp. *downwards*] *closed*, if $X' = \uparrow X'$ [resp. $X' = \downarrow X'$]. Let $\mathcal{F}(X)$ [resp. $\mathcal{I}(X)$] denote the set of all upwards [resp. downwards] closed non-empty subsets of X. It is natural to define the order on $\mathcal{I}(X)$ as subset inclusion, and the order on $\mathcal{F}(X)$ as *superset inclusion*.

If X and Y are sets, then $X + Y$ denotes their *disjoint union*. Formally, $X + Y = \{(1, x) \,|\, x \in X\} \cup \{(2, y) \,|\, y \in Y\}$, but if there is no chance of confusing the membership of the elements of $X + Y$ in X or Y, we will simply consider X and Y as subsets of $X + Y$. Similarly, if X is a set and $n \in \mathbb{N}$, then nX denotes the disjoint union of n copies of X, formally $nX = \{(i, x) \,|\, x \in X, i \in \mathbb{N}, 1 \leq i \leq n\}$.

A *(countably supported) probability distribution* over the set X is a mapping $\chi : X \to \mathbb{R}$ satisfying

- $\chi(x) \geq 0$ for all $x \in X$;
- The set $\{x \in X \,|\, \chi(x) > 0\}$, denoted $\underline{\chi}$, has at most countable cardinality;
- $\sum_{x \in X} \chi(x) = 1$.

We denote the set of all countably supported probability distributions over the set X by $\mathcal{D}(X)$. Any mapping $f : X \to Y$ can be lifted to $\mathcal{D}(f) : \mathcal{D}(X) \to \mathcal{D}(Y)$ by $\mathcal{D}(f)(\chi)(y) = \sum_{x \in X, f(x) = y} \chi(x)$ for all $\chi \in \mathcal{D}(X)$ and $y \in Y$. With such lifting, $\mathcal{D}(\cdot)$ is an endofunctor on the category of sets. It can be given the structure of a monad: the *unit* is the mapping $\eta : X \to \mathcal{D}(X)$ defined by $\eta(x)(y) = \delta_{xy}$. The *multiplication* is the mapping $\mu : \mathcal{D}(\mathcal{D}(X)) \to \mathcal{D}(X)$ defined for each $\mathcal{X} \in \mathcal{D}(\mathcal{D}(X))$ by $\mu(\mathcal{X})(x) = \sum_{\chi \in \mathcal{D}(X)} \mathcal{X}(\chi) \cdot \chi(x)$. Due to the monad structure, we can define the *Kleisli composition* of mappings $f : X \to \mathcal{D}(Y)$ and $g : Y \to \mathcal{D}(Z)$ by $g \circ_{\mathrm{Kl}} f = \mu \circ \mathcal{D}(g) \circ f : X \to \mathcal{D}(Z)$. Operationally, such composition works exactly as we imagine two probabilistic programs to be sequentially composed: f gets an argument x and probabilistically outputs a result y, on which g is invoked and the result z obtained, again probabilistically.

For $f : X \to \mathcal{D}(Y)$ we also use the notation $\overline{f} : \mathcal{D}(X) \to \mathcal{D}(Y)$ to denote the lifting of f to probability distributions. Formally, $\overline{f} = \mu \circ \mathcal{D}(f)$. The notions we have defined satisfy $g \circ_{\mathrm{Kl}} f = \overline{g} \circ f$.

Given a probability distribution $\psi \in \mathcal{D}(X \times Y)$, we let $\psi{\downarrow_1} \in \mathcal{D}(X)$ and $\psi{\downarrow_2} \in \mathcal{D}(Y)$ denote its projections to the first and second component, respectively. These are defined by $\psi{\downarrow_1}(x) = \sum_{y \in Y} \psi(x, y)$ for all $x \in X$ and similarly for $\psi{\downarrow_2}$, where the sum is well-defined due to the support of ψ being countable. In the other direction, given probability distributions $\chi \in \mathcal{D}(X)$ and $\phi \in \mathcal{D}(Y)$, we define the distribution $\chi \times \phi \in \mathcal{D}(X \times Y)$ by setting $(\chi \times \phi)(x, y) = \chi(x) \cdot \phi(y)$.

For a distribution $\rho \in \mathcal{D}(\mathbb{R})$, its *average* is defined by $\mathbb{E}[\rho] = \sum_{r \in \mathbb{R}} r \cdot \rho(r)$. Note that the sum only has a countable number of non-zero summands due to the support of ρ being at most countable. Nevertheless, the sum may diverge or be undefined. Defining the distribution $\rho^2 \in \mathcal{D}(\mathbb{R}_+)$ as $\rho^2(x) = \rho(\sqrt{x}) + \rho(-\sqrt{x})$, the *variance* of ρ is defined by $\mathbb{V}[\rho] = \mathbb{E}[\rho^2] - \mathbb{E}[\rho]^2$.

We are going to use *multiset comprehensions* to construct new probability distributions from existing ones. The write-up $\{\!|E \,|\, C|\!\}$ denotes the distribution of the values of the expression E, controlled by the

list of conditions C. This list contains the definitions of values through two mechanisms: computation and sampling from an already defined distribution, the latter denoted by $x \leftarrow \chi$. All samplings in the list of conditions are independent of each other; $x, x' \leftarrow \chi$ is shorthand for $x \leftarrow \chi, x' \leftarrow \chi$. E.g. the distribution ρ^2 above could be defined as $\{\!|x^2 \mid x \leftarrow \rho|\!\}$.

For $S : Y \to Z$ and $X \subseteq Y$, we let $S|_X$ denote the *restriction* of S (as a mapping) to the subset X. We extend this notion to probability distributions: for $D \in \mathcal{D}(Y \to Z)$, let $D|_X \in \mathcal{D}(X \to Z)$ denote the probability distribution defined by $D|_X(S') = \sum_{S:Y\to Z, S|_X = S'} D(S)$. In particular, for an element $y \in Y$, we let $D|_y \in \mathcal{D}(Z)$ denote the distribution of the value of variable y in D.

3.3.1.2 Differential Privacy.

Definition 3.1 (Metric space)**.** A *metric* space is a set X together with a *metric* d_X on it. A mapping $d_X : X \times X \to \mathbb{R}_+$ is a metric if it satisfies the following conditions:

- for all $x, y \in X$: $d_X(x, y) = 0$ iff $x = y$;
- for all $x, y \in X$, $d_X(x, y) = d_X(y, x)$;
- for all $x, y, z \in Z$, $d_X(x, z) \leq d_X(x, y) + d_X(y, z)$.

Let $\mathcal{R}$ be the set of possible database records and $X = \mathbb{N}^{\mathcal{R}}$ be the set of databases (i.e. a database is a multiset of records). Let O be a set of possible outcomes and $\mathcal{M} : X \to O$ a probabilistic map (an *information release mechanism*). For $r \in \mathcal{R}$ let $x_1 \overset{r}{\sim} x_2$ denote that x_1, x_2 *differ only by* r, i.e. $x_1(r) = x_2(r) \pm 1$ and $x_1(r') = x_2(r')$ for all $r' \in \mathcal{R}\backslash\{r\}$. Two databases $x_1, x_2 \in X$ are *adjacent* if $x_1 \overset{r}{\sim} x_2$ for some $r \in \mathcal{R}$. Let d_X be any metric on X.

Definition 3.2 (Differential privacy [36])**.** Let $\varepsilon \in \mathbb{R}$. The mechanism $\mathcal{M}$ is *ε-differentially private* if $\Pr[\mathcal{M}(x_1) \in S] \leq e^{\varepsilon} \cdot \Pr[\mathcal{M}(x_2) \in S]$ for all $S \subseteq O$ and all adjacent databases $x_1, x_2 \in X$.

Definition 3.3 (d_X-privacy [37])**.** The mechanism $\mathcal{M}$ is *d_X-private* if $\Pr[\mathcal{M}(x_1) \in S] \leq e^{d_X(x_1,x_2)} \cdot \Pr[\mathcal{M}(x_2) \in S]$ for all $S \subseteq O$ and $x_1, x_2 \in X$.

Definition 3.4 (Personalised differential privacy [38])**.** Let $\mathcal{E} : \mathcal{R} \to \mathbb{R}$. The mechanism $\mathcal{M}$ is *$\mathcal{E}$-differentially private* if $\Pr[\mathcal{M}(x_1) \in S] \leq e^{\mathcal{E}(r)} \cdot \Pr[\mathcal{M}(x_2) \in S]$ for all $S \subseteq O$, all $r \in \mathcal{R}$ and all databases $x_1, x_2 \in X$ satisfying $x_1 \overset{r}{\sim} x_2$.

Note that in Def. 3.3, the privacy level ε is implicitly part of d_X as a scale factor. Def. 3.3 presents a very general methodology to define the privacy of a mechanism, and Def. 3.4 gives a possible recipe to instantiate it. Indeed, for any $\mathcal{E} : \mathcal{R} \to \mathbb{R}$ there exists a metric $d_{\mathcal{E}}$, such that a mechanism $\mathcal{M}$ is $\mathcal{E}$-differentially private iff it is $d_{\mathcal{E}}$-private. A suitable construction of that metric is $d_{\mathcal{E}}(x_1, x_2) = \sum_{r\in\mathcal{R}} \mathcal{E}(r) \cdot |x_1(r) - x_2(r)|$ (for databases with finite support). We believe that the privacy definitions useful for the construction of a privacy analyzer will be similar to Def. 3.4: the input objects x consist of components, the distance between two objects is mostly determined by the components in which these objects differ, and privacy is defined as in Def. 3.3. In Sec. 3.3.6 we give a very general methodology for defining such distances.

Definition 3.5 (Sensitivity)**.** Let X and Y be two metric spaces with distances d_X and d_Y on them. Let $c \in \mathbb{R}_+$. We say that a function $f : X \to Y$ is *c-sensitive*, if for all $x_1, x_2 \in X$, the inequality $d_Y(f(x_1), f(x_2)) \leq c \cdot d_X(x_1, x_2)$ holds.

Sensitivity is the main tool in arguing the differential privacy of various information release mechanisms. For mechanisms that first compute a "useful" function and then add noise to it, the differential privacy of the resulting mechanism is the ratio of the sensitivity of that function and the magnitude of the added noise. Differential privacy itself can also be seen as an instance of sensitivity. Indeed, define the following distance d_{dp} over $\mathcal{D}(Y)$:

$$d_{\mathrm{dp}}(\chi, \chi') = \inf\{\varepsilon \in \mathbb{R}_+ \mid \forall y \in Y : e^{-\varepsilon}\chi'(y) \leq \chi(y) \leq e^{\varepsilon}\chi'(y)\}$$

or, equivalently

$$d_{\mathrm{dp}}(\chi, \chi') = \sup_{x \in X} \left| \ln \frac{\chi(x)}{\chi'(x)} \right| \quad .$$

Then a mechanism $\mathcal{M}$ from X to Y is d_X-private iff it is 1-sensitive with respect to the distances d_X on X and d_{dp} on $\mathcal{D}(Y)$.

The well-known composition theorems of differential privacy are instantiations of more general results on sensitivity of composed mappings. We start with the simplest result for sensitivity.

Proposition 3.1. *Let $f : X \to Y$ be c-sensitive with respect to the distances d_X on X and d_Y on Y. Let $f' : Y \to Z$ be c'-sensitive with respect to the distances d_Y on Y and d_Z on Z. Then $f' \circ f : X \to Z$ is $c \cdot c'$-sensitive with respect to the distances d_X on X and d_Z on Z.*

Proof. Let $x, x' \in X$. Then $d_Z(f'(f(x)), f'(f(x'))) \leq c' \cdot d_Y(f(x), f(x')) \leq c' \cdot c \cdot d_X(x, x')$. □

This proposition can be generalized to multivariate mappings. Let $i \in \{1, \ldots, n\}$. We say that a mapping $f' : Y_1 \times \cdots \times Y_n \to Z$ is *c_i'-sensitive in its i-th argument*, if for all tuples $(y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_n) \in Y_1 \times \cdots \times Y_{i-1} \times Y_{i+1} \times \cdots \times Y_n$, the univariate mapping $f(y_1, \ldots, y_{i-1}, \cdot, y_{i+1}, \ldots, y_n)$ is c_i'-sensitive.

Proposition 3.2. *For $i \in \{1, \ldots, n\}$, let $f_i : X \to Y_i$ be c_i-sensitive with respect to the distances d_X on X and d_{Y_i} on Y_i. Let $f' : Y_1 \times \cdots \times Y_n \to Z$ be c_i'-sensitive with respect to the distances d_{Y_i} on Y_i and d_Z on Z (for all $i \in \{1, \ldots, n\}$). Then the mapping $g : X \to Z$, defined by $g(x) = f'(f_1(x), \ldots, f_n(x))$, is $\sum_{i=1}^{n} c_i c_i'$-sensitive with respect to the distances d_X on X and d_Z on Z.*

Proof. Let $x, x' \in X$. Let $z_i = f'(f_1(x), \ldots, f_i(x), f_{i+1}(x'), \ldots, f_n(x'))$. Then $z_0 = g(x')$, $z_n = g(x)$ and by Prop. 3.1, $d_Z(z_{i-1}, z_i) \leq c_i c_i' \cdot d_X(x, x')$. The claim of the proposition follows from the triangle inequality. □

Proposition 3.3 (Sequential composition for DP)**.** *Let $f : X \to Y$ be c-sensitive with respect to the distances d_X on X and d_Y on Y. Let $a : Y \to Z$ be* 1*-sensitive with respect to the distances d_Y on Y and d_Z on Z. Let $g : X \times Z \to W$ be such that for all $z \in Z$, $g(\cdot, z)$ is c'-sensitive with respect to the distances d_X on X and d_W on W. Also, for all $x \in X$, let $g(x, \cdot)$ be* 1*-sensitive with respect to the distances d_Z on Z and d_W on W. Then the function $h : X \to W$, defined by $h(x) = g(x, a(f(x)))$ is $(c + c')$-sensitive with respect to the distances d_X on X and d_W on W.*

Proof. Let $x, x' \in X$. Then

$$\begin{aligned} d_W(h(x), h(x')) = d_W(g(x, a(f(x))), g(x', a(f(x')))) \leq \\ d_W(g(x, a(f(x))), g(x, a(f(x')))) + d_W(g(x, a(f(x'))), g(x', a(f(x')))) \leq \\ c \cdot d_X(x, x') + c' \cdot d_X(x, x') \quad . \end{aligned}$$ □

In the setting of differential privacy, there is a dataset $x \in X$ and information release mechanisms $\mathcal{M}_1$ and $\mathcal{M}_2$, which are respectively ε_1- and ε_2-differentially private. Let the possible set of outcomes of $\mathcal{M}_i$ be M_i. First $\mathcal{M}_1$ and then $\mathcal{M}_2$ are invoked on x; the exact invocation of $\mathcal{M}_2$ may depend on the result of $\mathcal{M}_1$. Finally, the result of $\mathcal{M}_2$ is published. This result may include the result of $\mathcal{M}_1$, because it affected the invocation of $\mathcal{M}_2$. Such composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ is shown to be $(\varepsilon_1 + \varepsilon_2)$-differentially private.

Proposition 3.3 is applicable to this case in the following manner.

- The set X corresponds to the set X in the proposition.
- $\mathcal{M}_1$ is the mapping f in the proposition. The set $\mathcal{D}(M_1)$ of probability distributions corresponds to the set Y, with d_{dp} over $\mathcal{D}(M_1)$ taking the role of d_Y.

- The mapping a in the proposition corresponds to any further processing done to the result of $\mathcal{M}_1$ before using it to guide the workings of $\mathcal{M}_2$. This processing is assumed to be deterministic, with the output in some set M'. But as $\mathcal{M}_1$ gave its output probabilistically, such processing is also lifted to a mapping from $\mathcal{D}(M_1)$ to $\mathcal{D}(M')$. The set $\mathcal{D}(M')$ corresponds to the set Z in the proposition, with d_{dp} taking the role of d_Z. As no postprocessing can worsen the privacy guarantees of a differentially private mechanism, the sensitivity of a is at most 1.
- The mapping f' in the proposition corresponds to the mechanism $\mathcal{M}_2$; its second argument is the guidance it receives from $\mathcal{M}_1$. The set $\mathcal{D}(M_2)$ corresponds to the set W in the theorem, with d_{dp} taking the role of d_W. Again, $g(x, \cdot)$ is at most 1-sensitive, because postprocessing does not weaken the privacy guarantees.

The set-up of the (ill-named) parallel composition is more complex. There are the same mechanisms $\mathcal{M}_1$ and $\mathcal{M}_2$ applied to $x \in X$ *one after the other*, with the invocation of $\mathcal{M}_2$ depending on the result of $\mathcal{M}_1$. Additionally, it is stated that $\mathcal{M}_1$ and $\mathcal{M}_2$ "consider only independent parts of the dataset x". In the original treatment, the "independent parts" were intended to be e.g. different columns of a database. In general, however, a "part" of $x \in X$ can be modeled as an equivalence relation over X. It relates two elements of X if they are equal to each other in the considered "part". An information release mechanism makes use of only that "part" if it is equal on related elements. Hence we have equivalence relations ρ_1, ρ_2, such that $\mathcal{M}_i(x) = \mathcal{M}_i(x')$ if $x \ \rho_i \ x'$. An additional condition states that the "parts" have to be independent. This can be clearly interpreted as a condition involving the equivalence relations ρ_1, ρ_2 and the distance on X. We state a suitable condition in Prop. 3.4 below; it is more general than what is used in the argument of parallel composition for DP.

Proposition 3.4 (Parallel composition for DP)**.** *Let $f : X \to Y$, $a : Y \to Z$, $g : X \times Z \to W$ have the same sensitivities as in Prop. 3.3. Let ρ, σ be equivalence relations on X that are* independent*, i.e. for any x, x', there exist $m \in \mathbb{N}$ and $x_0, \ldots, x_{2m} \in X$, such that $\sum_{i=1}^{2m} d_X(x_{i-1}, x_i) = d_X(x, x')$ and*

$$x = x_0 \ \rho \ x_1 \ \sigma \ x_2 \ \cdots \rho \ x_{2m-1} \ \sigma \ x_{2m} = x' \ . \tag{2}$$

For each $x, x' \in X$ and $z \in Z$, let $x \ \rho \ x'$ imply $f(x) = f(x')$, and $x \ \sigma \ x'$ imply $g(x, z) = g(x', z)$. Then the function $h : X \to W$, defined by $h(x) = g(x, a(f(x)))$ is $\max\{c, c'\}$-sensitive with respect to the distances d_X on X and d_W on W.

Proof. Let $x, x' \in X$. Let m and $x_0, \ldots, x_{2m}$ be as in (2). Then

$$d_W(h(x), h(x')) = d_W(g(x, a(f(x))), g(x', a(f(x')))) \leq \left(\sum_{i=1}^{2m} d_W(g(x_{i-1}, a(f(x_{i-1}))), g(x_i, a(f(x_i)))) \right) =$$
$$\left(\sum_{i=1}^{m} d_W(g(x_{2i-2}, a(f(x_{2i-2}))), g(x_{2i-1}, a(f(x_{2i-1})))) \right) + \left(\sum_{i=1}^{m} d_W(g(x_{2i-1}, a(f(x_{2i-1}))), g(x_{2i}, a(f(x_{2i})))) \right) =$$
$$\left(\sum_{i=1}^{m} d_W(g(x_{2i-2}, a(f(x_{2i-2}))), g(x_{2i-1}, a(f(x_{2i-2})))) \right) + \left(\sum_{i=1}^{m} d_W(g(x_{2i}, a(f(x_{2i-1}))), g(x_{2i}, a(f(x_{2i})))) \right) \leq$$
$$\left(\sum_{i=1}^{m} c' \cdot d_X(x_{2i-2}, x_{2i-1}) \right) + \left(\sum_{i=1}^{m} c \cdot d_X(x_{2i-1}, x_{2i}) \right) \leq \max\{c, c'\} \cdot \sum_{i=1}^{2m} d_X(x_{i-1}, x_i) = \max\{c, c'\} \cdot d_X(x, x') \ .$$

Here the equality between second and third row follows from

- (for left summand) $x_{2i-2} \ \rho \ x_{2i-1}$ by (2). Hence $f(x_{2i-1}) = f(x_{2i-2})$.
- (for right summand) $x_{2i-1} \ \sigma \ x_{2i}$ by (2). Hence $g(x_{2i-1}, z) = g(x_{2i}, z)$ for $z = a(f(x_{2i-1}))$.

□

3.3.2 Sensitivity of Data Processing Workflows. The sensitivity analysis for data processing workflows (DP-workflows) considers workflows made up of tasks and data objects. Tasks create and consume data objects; each data object is created and consumed by a single task. Each task has a sensitivity, which has already been determined. Our analysis will determine the sensitivity of the entire workflow.

3.3.2.1 DP-Workflows. More concretely, a DP-Workflow is a directed acyclic graph (DAG) where each node represents either a data node or a (DP-)task, and the arcs go either from a data node to a task node (input) or from a task node to a data node (output). Additionally, because there is no conditionally branching or merging, no two tasks in the corresponding DP-Workflow can share the same output data node (i.e. a data node is output of at most one task).

A data processing workflow consists of *data nodes*, *processing nodes* and *data-flow arcs*. A data-flow arc connects a data node to a processing node or vice-versa. A data node without any incoming arc is called a *source data node*. It corresponds to an object or collection of objects that are given as input to the workflow. A data node without any outgoing arc is called an *output node* (i.e. it is data produced by an execution of the workflow). A data node with both incoming and outgoing arcs is called an *intermediate node*.

Figure 31 shows an example of a data processing workflow. Data nodes are represented as rounded rectangles, while data nodes are rectangles with their top-right corner folded over.

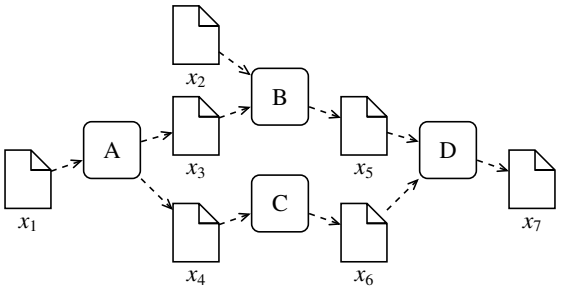


Figure 31: Example of a Data Processing Workflow (DP-Workflow)

Formally, a *Data Processing Workflow* W is a tuple (D, P, F), where D and P are two finite, disjoint sets, and F is a relation on $(D \times P) \cup (P \times D)$. For convenience, we will refer to $D \cup P$ as the set of nodes N. The elements of D are data nodes and the elements of P are processing nodes, that is, nodes representing computations over some input data.

Given a node $n \in N$, we define $\bullet n = \{m \mid (m, n) \in F\}$ (the predecessors of n) and $n\bullet = \{m \mid (n, m) \in F\}$ (the successors of n). A DP-Workflow W is said *well-formed* if it induces an acyclic, weakly connected graph, with the following additional restrictions: every node $d \in D$ has at most one successor and at most one predecessor, i.e. $| \bullet d | \leq 1$ and $| d\bullet | \leq 1$, and every node $p \in P$ has at least one predecessor and at least one successor node, i.e. $| \bullet p | \geq 1$ and $| p\bullet | \geq 1$. In the following, we consider only well formed DP-Workflows.

A privacy-enhanced DP-Workflow is a DP-Workflow annotated with differential privacy and sensitivity values. Formally, a *Privacy-enhanced DP-Workflow* is a tuple $(W, \mathcal{E}, C)$, where $W = (D, P, F)$ is a DP-Workflow and $\mathcal{E}$ and C are mappings $D \times P \times D \to \mathbb{R}_+$, associating a differential privacy and sensitivity value (respectively) to an output produced by a processing node, relative to an input of this processing node.

For example, a privacy-enhanced version of the DP-Workflow shown in Figure 31 is shown in Figure 32. In the figure, we use $\epsilon_A[x_1, x_3] = 0.2$ to denote the tuple $(x_1, A, x_3, 0.2) \in \mathcal{E}$, meaning that performing A is ϵ-differential private with $\epsilon = 0.2$, when processing x_1 as input and producing x_3. Similarly, $c_A[x_1, x_3] = 0.4$ is used to denoted the tuple $(x_1, A, x_3, 0.4) \in C$, which means that A takes as input x_1 and produces x_3 with a sensitivity of 0.4.

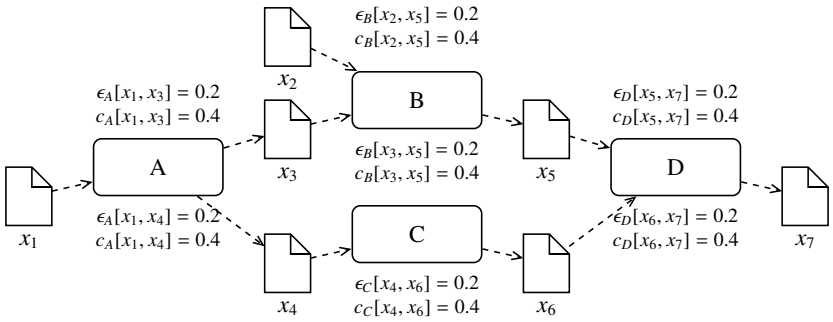


Figure 32: Example of a Privacy-Enhanced DP-Workflow

3.3.2.2 Data Node-Based Analysis of DP-Workflows. We are interested in computing upper bounds of the information disclosed when data nodes are accessed by a user playing a given role, or sent out to an external party. In order to do so, we design an algorithm that computes the differential privacy and sensitivity values of every intermediate and output data node in a privacy enhanced DP-Workflow, relative to every source data node. Subsequently, we show how to aggregate the privacy and sensitivity values calculated in this way, in order to compute a bound of the information that each role or external party can extract from each source data node, given the data that are disclosed to them during one execution of the workflow.

The proposed algorithm is given below. The input of the algorithm is a privacy-enhanced DP-workflow, while the output consists of two matrices, namely d_{dp} and d_c, of size $|S| \times |O|$ where S is the set of source data nodes in the workflow and O is the set of intermediate and output data nodes. A cell in d_{dp} (respectively d_c) gives a differential privacy bound (resp. sensitivity bound) of a given intermediate or output data node relative to a source data node. The main idea of the algorithm is to iterate over the processing nodes in the DP-Workflow in topological order (which requires that the DP-Workflow is well-formed and thus acyclic). At each step, we compute the value of $d_{dp}[s, o]$ and $d_c[s, o]$ for each output o of the current processing node p, using the previously computed values for the input data nodes of p, as well as the formulas for composing sensitivity values given in Propositions 3.1–3.4 and existing formulas for composition of $\epsilon-$differentially private information release mechanisms.

Data: A well-formed *DP-Workflow* (W, S), with $W = (D, P, F)$
Result: The matrices d_{dp} and d_c
foreach processing node $p \in P$ in topological order **do**
  foreach $s \in D, o \in p\bullet : |\bullet s| = \overline{0 \wedge (s, o) \in F^+}$ **do**
    if $s \in \bullet p$ **then**
      $d_{dp}[s, o] = \epsilon_p[s, o]$
      $d_c[s, o] = c_p[s, o]$
    else
      $d_{dp}[s, o] = \sum_{i \in \bullet p : (s,i) \in F^+} \min\left(d_{dp}[s, i],\ d_c[s, i] \cdot \epsilon_p[i, o]\right)$
      $d_c[s, o] = \sum_{i \in \bullet p : (s,i) \in F^+} \left(d_c[s, i] \cdot c_p[i, o]\right)$
    end
  end
end
return d_{dp}, d_c

Algorithm 1: Differential Privacy of a DP-Workflow

Example 3.1. We use the example in Figure 32 to illustrate the algorithm. To this end, we consider the

topological order $[A, B, C, D]$ of processing nodes[9].

During the first iteration, in line 1 the algorithm sets p to the processing node A. In line 2, the algorithm iteratively selects a source data node (i.e. $s \in D : |\bullet s| = 0$) and one successor of p such that the latter is reachable from the selected source node. The first iteration of the inner loop then processes the pair $s = x_1$ and $o = x_3$. Since x_1 is a direct predecessor of A the algorithm will perform lines 4-5. As a result, we have that $d_{dp}[x_1, x_3] = \epsilon_A[x_1, x_3] = 0.2$ and $d_c[x_1, x_3] = c_A[x_1, x_3] = 0.4$. The second iteration of the inner loop, in turn, will process the pair $s = x_1$ and $o = x_4$. The latter will result in $d_{dp}[x_1, x_4] = \epsilon_A[x_1, x_4] = 0.2$ and $d_c[x_1, x_4] = c_A[x_1, x_4] = 0.4$. This will complete the first iteration of the outer loop because none of the successors of A is reachable from x_2. The following matrices summarize the outcome of the first iteration:

	x_3	x_4	x_5	x_6	x_7
x_1	$\epsilon_A[x_1, x_3] = 0.2$	$\epsilon_A[x_1, x_4] = 0.2$			
x_2					

d_{dp}

	x_3	x_4	x_5	x_6	x_7
x_1	$c_A[x_1, x_3] = 0.4$	$c_A[x_1, x_4] = 0.4$			
x_2					

d_c

In the second iteration, the algorithm sets p to the processing node B (line 1). The inner loop first computes the values for source node x_1 and the only successor of B, that is x_5. This time, the algorithm executes lines 7-8, because x_1 is not a direct predecessor of B. Note that x_3 is the only direct predecessor of B which is reachable from x_1 and, as a result, there is only one term in the summation of line 7. Therefore, in line 7 we have that $d_{dp}[x_1, x_5] = \min(d_{dp}[x_1, x_3], d_c[x_1, x_3] \cdot \epsilon_B[x_3, x_5]) = \min(0.2, 0.4 \cdot 0.2) = 0.08$ and in line 8 $d_c[x_1, x_5] = d_c[x_1, x_3] \cdot c_B[x_3, x_5] = 0.4 \cdot 0.4 = 0.16$. In the second iteration of the inner loop, the algorithm computes the values associated to the source node x_2 and the only successor of b, that is x_5. Since x_2 is direct predecessor of B, the algorithm sets $d_{dp}[x_2, x_5] = \epsilon_B[x_2, x_5] = 0.2$ and $d_c[x_2, x_5] = c_B[x_2, x_5] = 0.4$.

The third iteration selects $p = C$ and proceeds in a similar way as for the second iteration. The following matrices summarize the values computed at the end of this iteration.

	x_3	x_4	x_5	x_6	x_7
x_1	0.2	0.2	0.08	0.08	
x_2			0.2		

d_{dp}

	x_3	x_4	x_5	x_6	x_7
x_1	0.4	0.4	0.16	0.16	
x_2			0.4		

d_c

In the final iteration, the algorithm computes the values by selecting p to be the processing node D. In the inner loop, the algorithm will first select the source node x_1. Note that D has x_7 as its only successor. However, there are two direct predecessors of D, namely x_5 and x_6. Therefore the computation of d_{dp} involves the summation of the values that come from x_5 and x_6. Thus, we have that:

$$\begin{aligned} d_{dp}[x_1, x_7] &= \min\left(d_{dp}[x_1, x_5], d_c[x_1, x_5] \cdot \epsilon_D[x_5, x_7]\right) + \\ &\quad \min\left(d_{dp}[x_1, x_6], d_c[x_1, x_6] \cdot \epsilon_D[x_6, x_7]\right) \\ &= \min(0.08, 0.16 \cdot 0.2) + \min(0.08, 0.16 \cdot 0.2) \\ &= 0.064 \end{aligned}$$

and

$$\begin{aligned} d_c[x_1, x_7] &= (d_c[x_1, x_5] \cdot c_D[x_5, x_7]) + (d_c[x_1, x_6] \cdot c_D[x_6, x_7]) \\ &= (0.16 \cdot 0.4) + (0.16 \cdot 0.4) \\ &= 0.128 \end{aligned}$$

In the final iteration of the inner loop, the algorithm computes the values for $s = x_2$ and $o = x_5$. In this case however, there is only one term in the summation. Therefore, $d_{dp}[x_2, x_7] = \min\left(d_{dp}[x_2, x_5], d_c[x_2, x_5] \cdot \epsilon_D[x_5, x_7]\right) = \min(0.2, 0.4 \cdot 0.2) = 0.08$. Finally, $d_c[x_2, x_7] = d_c[x_2, x_5] \cdot c_D[x_5, x_7] = 0.4 \cdot 0.4 = 0.16$.

The following matrices summarize the outcome of the algorithm.

[9]Note that there exists another topological order of the processing nodes of the example, namely $[A, C, B, D]$. Either one would produce the same output matrices.

	x_3	x_4	x_5	x_6	x_7
x_1	0.2	0.2	0.08	0.08	0.064
x_2			0.2		0.16

d_{dp}

	x_3	x_4	x_5	x_6	x_7
x_1	0.4	0.4	0.16	0.16	0.128
x_2			0.4		0.08

d_c

3.3.2.3 Role-Based Privacy Analysis of DP-Workflows. The disclosures identified in a PE-BPMN process model can be encoded in a relation $Disc \subseteq D \times R$, such that $Disc(n, r)$ denotes the fact that data node n is disclosed to role r. Given the matrices d_{dp} and d_c computer from a privacy-enhanced DP-Workflow W and given the relation $Disc$ capturing the disclosure of data nodes to roles, we can now compute a differential privacy bound $\epsilon_r(s)$ of the information that a given role r can extract from a given source data node s – i.e. how much a party playing a given role can learn about individual records of a given input s of W:

$$\epsilon_r(s) = \sum_{(n,r) \in Disc \,:\, (s,n) \in F^+} d_{dp}[s, n] \tag{3}$$

Example 3.2. Given the matrices computed in the previous example:

	x_3	x_4	x_5	x_6	x_7
x_1	0.2	0.2	0.08	0.08	0.064
x_2			0.2		0.16

d_{dp}

	x_3	x_4	x_5	x_6	x_7
x_1	0.4	0.4	0.16	0.16	0.128
x_2			0.4		0.08

d_c

we can compute the differential privacy guarantee with respect to data node x_1 that can be made for a party playing a role r that has access to both data nodes x_5 and x_6 in the DP-Workflow shown in Figure 32:

$$\begin{aligned} \epsilon_r(x_1) &= d_{dp}[x_1, x_5] + d_{dp}[x_1, x_6] \\ &= 0.08 + 0.08 = 0.16 \end{aligned}$$

In the above equation and example we sum up the ϵ values calculated for each intermediate/output data node that is disclosed to role r. This is a worst-case bound. If a role has access to two data nodes n_1 and n_2 produced from a given source data node s via two different paths in the DP-Workflow (thus implying at least partially different information release mechanisms), it would be possible to derive a tighter bound – in the best case $\max(d_{dp}[s, n_1], d_{dp}[s, n_2])$ instead of $d_{dp}[s, n_1] + d_{dp}[s, n_2]$. Such refinements of the bound will be investigated in the next six-months iteration of the project.

3.3.3 Sensitivity and DP Analysis of Components. Above, we gave a sensitivity and differential privacy analysis for workflows, making use of the already-computed values for sensitivity (and DP) for individual processing nodes. As next, we present an analysis for the nodes, the operations of which have been specified. In this presentation, the specification language is a simple programming language. The language is geared towards constructing differentially private data transformers, containing primitive operations for inserting noise generated according to useful distributions.

3.3.3.1 Programming Language. Let **Var** be the set of variables. The *arithmetic expressions* e, *boolean expressions* b and *programs* s are defined by the grammar given in Fig. 33.

A *state* of the program maps variables to real numbers[10], let **State** = **Var** $\to \mathbb{R}$ be the set of all program states. For $x \in$ **Var**, $v \in \mathbb{R}$ and $S \in$ **State** let $S' = S[x \mapsto v]$ denote a state that satisfies $S'(x) = v$ and $S'(y) = S(y)$ for $y \in$ **Var**$\backslash\{x\}$.

[10]If the potential values really were real numbers, then the set of states would be non-countable and formally defining probability distributions over them would be significantly more complicated. In our discussions we assume that $\mathbb{R}$ has been discretized sufficiently well, such that there are only countably many values, but on the other hand, noise can be generated with sufficient precision.

We have simplified the language by leaving out looping constructs as well as data structures of arbitrary size. The main reason for leaving them out is our lack of good analysis methods for them. But we provide a partial solution in Sec. 3.3.3.6.

The semantics of a program s has the type $[\![s]\!] : \mathbf{State} \to \mathcal{D}(\mathbf{State})$. It's definition is given in Fig. 34 together with the semantices for arithmetic expressions — $[\![e]\!] : \mathbf{State} \to \mathcal{D}(\mathbb{R})$ — and boolean expressions — $[\![b]\!] : \mathbf{State} \to \mathcal{D}(\mathbb{B})$. The definitions are straightforward due to the lack of possibilities for the programs to diverge.

3.3.3.2 Abstract Interpretation. Abstract interpretation [39] is a general program analysis technique consisting of defining an *abstract* semantics for the programming language that faithfully abstracts the concrete semantics, and computing this semantics for the program we want to analyze. A good abstract semantics clearly expresses the properties of the programs we are interested in. The computation of the abstract semantics and the subsequent extraction of interesting properties should be more tractable than similar computation and extraction for the concrete semantics. In our case, the concrete semantics is probabilistic and we are interested in the distance d_{dp} for certain probability distributions extracted from the semantics of the program. We cannot think of any reasonable methods for extracting these distances directly from the actual executions of the programs we study.

To use abstract interpretation, we have to define the *abstract domain*, and the *transfer functions* for all possible program steps. Together, they form the abstract semantics of the language. The abstract domain **AState** is a partially ordered set. Each element of **AState** corresponds to some concrete states, i.e. elements of $\mathcal{D}(\mathbf{State})$. The correspondence is given by a relation $\vDash \subseteq \mathbf{AState} \times \mathcal{D}(\mathbf{State})$; this relation has to be specified together with **AState**. The order on **AState** is one of information: if $A_1, A_2 \in \mathbf{AState}$, $D \in \mathcal{D}(\mathbf{State})$ and $A_1 \leq A_2$, then $A_1 \vDash D$ implies $A_2 \vDash D$. This means that smaller elements of **AState** convey more precise information about the concrete executions of the program; there is less uncertainty in them. In typical analyses, the domain **AState** has the least element $\bot$ and the greatest element $\top$ satisfying $\bot \nvDash D$ and $\top \vDash D$ for all $D \in \mathcal{D}(\mathbf{State})$.

The transfer functions $[\![s]\!]_{\mathrm{A}}$ have the type $\mathbf{AState} \to \mathbf{AState}$. They must satisfy the following condition:

$$\forall A \in \mathbf{AState}, D \in \mathcal{D}(\mathbf{State}) : A \vDash D \Longrightarrow [\![s]\!]_{\mathrm{A}}(A) \vDash \overline{[\![s]\!]}(D) \ . \tag{4}$$

This condition means, that from a valid abstraction of the initial distribution of program states we must obtain some valid abstraction of the final distribution of states.

We note that composing the transfer functions of subprograms is the obvious way to define the transfer function for the sequential composition of programs. As the concrete semantics of composition is defined in terms of $\circ_{\mathrm{Kl}}$, this also satisfies (4).

Abstract interpretation for information flow. As described, abstract interpretation is well-suited to automatically derive properties satisfied by program states and distributions over them, e.g. the values of variables in the final state of the program. Unfortunately, information flow is not such a *trace property*. Instead of being defined over a single state or program execution, it is concerned with two different executions that start from states related in particular way. If we are interested in the differential privacy properties of the program, then we consider two executions from starting states with a certain distance

$$\begin{array}{rcl} e & ::= & x \\ & | & n \\ & | & e_1 + e_2 \\ & | & e_1 \cdot e_2 \\ & | & \mathit{Lap}() \end{array} \qquad \begin{array}{rcl} b & ::= & e_1 \leq e_2 \\ & | & e_1 = e_2 \\ & | & b_1 \wedge b_2 \\ & | & \neg b \end{array} \qquad \begin{array}{rcl} s & ::= & x := e \\ & | & \mathit{skip} \\ & | & s_1 ; s_2 \\ & | & \mathsf{if}\ b\ \mathsf{then}\ s_1\ \mathsf{else}\ s_2 \end{array}$$

Here $x \in \mathbf{Var}$ and $n \in \mathbb{R}$.

Figure 33: Syntax of the Programming Language

$$
\begin{aligned}
[\![x]\!](S) &= \eta(S(x)) \\
[\![n]\!](S) &= \eta(n) \\
[\![e_1 + e_2]\!](S) &= \{\!|v_1 + v_2 \mid v_1 \leftarrow [\![e_1]\!](S), v_2 \leftarrow [\![e_2]\!](S)|\!\} \\
[\![e_1 \cdot e_2]\!](S) &= \{\!|v_1 v_2 \mid v_1 \leftarrow [\![e_1]\!](S), v_2 \leftarrow [\![e_2]\!](S)|\!\} \\
[\![Lap()]\!](S) &= \mathrm{Lap}^1
\end{aligned}
$$

$$
\begin{aligned}
[\![e_1 \leq e_2]\!](S) &= \{\!|(v_1 \leq v_2)? \mid v_1 \leftarrow [\![e_1]\!](S), v_2 \leftarrow [\![e_2]\!](S)|\!\} \\
[\![e_1 = e_2]\!](S) &= \{\!|(v_1 = v_2)? \mid v_1 \leftarrow [\![e_1]\!](S), v_2 \leftarrow [\![e_2]\!](S)|\!\} \\
[\![b_1 \wedge b_2]\!](S) &= \{\!|v_1 \wedge v_2 \mid v_1 \leftarrow [\![b_1]\!](S), v_2 \leftarrow [\![b_2]\!](S)|\!\} \\
[\![\neg b]\!](S) &= \{\!|\neg v \mid v \leftarrow [\![b]\!](S)|\!\}
\end{aligned}
$$

$$
\begin{aligned}
[\![x := e]\!](S) &= \{\!|S[x \mapsto v] \mid v \leftarrow [\![e]\!](S)|\!\} \\
[\![skip]\!](S) &= \eta(S) \\
[\![s_1; s_2]\!](S) &= ([\![s_2]\!] \circ_{\mathrm{Kl}} [\![s_1]\!])(S) \\
[\![\textsf{if } b \textsf{ then } s_1 \textsf{ else} s_2]\!](S) &= \{\!|S''' \mid v \leftarrow [\![b]\!](S), S' \leftarrow [\![s_1]\!](S), S'' \leftarrow [\![s_2]\!](S), S''' = \begin{cases} S', & v \\ S'', & \neg v \end{cases}|\!\}
\end{aligned}
$$

Figure 34: Semantics of the Programming Language

apart, and ask whether the final distributions (or certain projections of them) have a not-too-large DP-distance.

Nevertheless, information flow is still relatively easily reduced to trace properties [40]. Given a program s, we consider the program $s; s'$, where s' is again the program s, but with each variable $x \in \mathbf{Var}$ replaced with a new variable $x' \in \mathbf{Var}'$. When we execute $s; s'$, we are effectively executing s twice, as long as it does not diverge (we have excluded divergence by removing loops from our language). Hence we can start $s; s'$ in a state where the values of the variables x and x' are related in a certain manner, and study the projections of the final distribution of states to variables in $\mathbf{Var}$ and to variables in $\mathbf{Var}'$.

Composition of abstractions. The abstraction may be a multi-step construction. Perhaps the transfer functions $[\![s]\!]_{\mathrm{A}}$ are also too difficult to evaluate. We may then define another abstract domain $\mathbf{AState}^\flat$, the relation $\vDash^\flat \subseteq \mathbf{AState}^\flat \times \mathbf{AState}$, the transfer functions $[\![s]\!]^f_{\mathrm{A}} lat$, and require them to abstractly interpret $\mathbf{AState}$ and $[\![s]\!]_{\mathrm{A}}$. In this case $\mathbf{AState}^\flat$ and $[\![s]\!]^\flat_{\mathrm{A}}$ is also an abstract interpretation of the concrete semantics through the relation $\vDash^\flat \cdot \vDash$.

3.3.3.3 Analysing Straight-line Programs. For now, let us simplify the the programming language even more, leaving out the *if*-statements as well. This gives us a program consisting entirely of assignments.

Abstract Domain. Let $\mathbf{Exp}(V)$ be the set of all expressions e and b as defined in Fig. 33, where

- the variables occurring there come from the set V;
- the construction *Lap*() is excluded.

In other words, $\mathbf{Exp}(V)$ is the set of all *deterministic* expressions over the variables in V. Let $\approx$ be an equivalence relation on $\mathbf{Exp}(V)$, defined by $e \approx e'$ iff $[\![e]\!](S) = [\![e']\!](S)$ for all $S : V \to \mathbb{R}$ (here false is equated with 0 and true with 1). We define the abstract state of programs as $\mathbf{AState} = \mathbf{AState}_1 \times$

$\mathbf{AState}_2 \times \mathbf{AState}_3 \times \mathbf{AState}_4$, where $\mathbf{AState}_i = \mathbf{AS}_i(\mathbf{Var})$ and

$$\begin{aligned}
\mathbf{AS}_1(V) &= \mathcal{P}(V) \\
\mathbf{AS}_2(V) &= \mathcal{P}(3V + \mathcal{P}(V^2) \to \mathbb{R}) \\
\mathbf{AS}_3(V) &= \mathcal{P}((\mathcal{P}(V))^2) \\
\mathbf{AS}_4(V) &= \mathcal{P}(V \times \mathbf{Exp}(V))
\end{aligned}$$

for any set V whose elements we interpret as variables. We also denote $\mathbf{AS}(V) = \mathbf{AS}_1(V) \times \mathbf{AS}_2(V) \times \mathbf{AS}_3(V) \times \mathbf{AS}_4(V)$.

Clearly, the abstract domain is too large for our program analysis to store its elements. We use this abstract domain to specify transfer functions in a clear manner, but our program analysis is actually going to implement a further abstraction of them, as discussed in Sec. 3.3.3.8.

In order to argue about the correctness of the transfer functions according to (4), we have to define the relation $\vDash$, stating which distributions of concrete states match an abstract state. We first state the following *abstraction function* $\alpha_2 : \mathcal{D}(\mathbf{State}) \to \mathbf{AState}_2$:

$$\begin{aligned}
\alpha_2(D) \quad = \quad & \{f : 3\mathbf{Var} + \mathcal{P}(\mathbf{Var}^2) \to \mathbb{R} \mid S \in \underline{D}, f((1,x)) = S(x), f((2,x)) = \mathbb{E}[D|_x], f((3,x)) = \mathbb{V}[D|_x], \\
& f(\{(x_1,y_1),\dots,(x_k,y_k)\}) = d_{\mathrm{dp}}(\{\!|(S(x_1),\dots,S(x_k)) \mid S \leftarrow D|\!\}, \\
& \qquad\qquad\qquad\qquad \{\!|(S(y_1),\dots,S(y_k)) \mid S \leftarrow D|\!\})\}
\end{aligned}$$

The elements f of $\alpha_2(D)$ can and should be interpreted as points in a vector space over $\mathbb{R}$, with the number of dimensions of the space being $3 \cdot |\mathbf{Var}| + 2^{|\mathbf{Var}|^2}$. The points describe the possible values of variables in states distributed according to D, as well as important details of the distribution D itself: the average and the variance of each variable. Also, the points record the DP-distances between various projections of the current distribution of program states.

We also state the following abstraction function $\alpha_3 : \mathcal{D}(\mathbf{State}) \to \mathbf{AState}_3$:

$$\alpha_3(D) = \{(X,Y) \mid X, Y \subseteq \mathbf{Var}, \{\!|(S|_X, S|_Y) \mid S \leftarrow D|\!\} = \{\!|(S|_X, S'|_Y) \mid S, S' \leftarrow D|\!\}\} \ .$$

We see that $\alpha_3(D)$ describes, which sets of variables are independent of each other in the distribution D.

Furthermore, the abstraction function $\alpha_4 : \mathcal{D}(\mathbf{State}) \to \mathbf{AState}_4$ is the following:

$$\alpha_4(D) = \{(x,e) \mid x \in \mathbf{Var}, e \in \mathbf{Exp}(\mathbf{Var}), \forall S \in \underline{D} : S(x) = [\![e]\!](S)\},$$

recording all relationships between the values of variables (of a special form) that hold for all states in the support of D.

Let $A_1 \in \mathbf{AState}_1$, $A_2 \in \mathbf{AState}_2$, $A_3 \in \mathbf{AState}_3$, $A_4 \in \mathbf{AState}_4$ and $D \in \mathcal{D}(\mathbf{State})$. We define $(A_1, A_2, A_3, A_4) \vDash D$, if $A_1 \vDash D$, $A_2 \vDash D$, $A_3 \vDash D$ and $A_4 \vDash D$, where

$$\begin{aligned}
A_1 \vDash D &\Leftrightarrow \forall x \in A_1 : D|_x \text{ is a Laplace distribution} \\
A_2 \vDash D &\Leftrightarrow \alpha_2(D) \subseteq A_2 \\
A_3 \vDash D &\Leftrightarrow \alpha_3(D) \supseteq A_3 \\
A_4 \vDash D &\Leftrightarrow \alpha_4(D) \supseteq A_4 \ .
\end{aligned}$$

Transfer Functions. The transfer functions of the analysis map the abstract state before an assignment $x := e$ to the state after it. In order to simplify the presentation, we restrict the arithmetic expressions in a way that obviously does not lessen the generality. Namely, we require that they contain a single arithmetic expression, i.e. the assignment statements match the following grammar:

$$x := x' | x := n | x := x_1 + x_2 | x := x1 \cdot x_2 | x := Lap() | x := (x_1 \le x_2)? | x := (x_1 = x_2)? \tag{5}$$

We demand that the variables in the right hand side of the assignments are different from x.

We have added the boolean expressions, in order to use them in Sec. 3.3.3.4. The boolean expressions are expected to return either 0 or 1. In this way, the conjunction and negation of expressions are simply arithmetic operations.

The analysis of an assignment in (5) proceeds in two steps: *killing* the left hand side and *generating* the right hand side. Let V be a set of variables, $x \in V$ and $V' = V \backslash \{x\}$. Let $(A_1, A_2, A_3, A_4) \in \mathbf{AS}(V)$. The killing of x is a mapping from $\mathbf{AS}(V)$ to $\mathbf{AS}(V')$, defined as follows:

$$kill_x(A_1, A_2, A_3, A_4) = \\ (A_1 \backslash \{x\}, \{f|_{3V' + \mathcal{P}((V')^2)} \mid f \in A_2\}, \{(X, Y) \in A_3 \mid x \notin X \cup Y\}, \{(y, e) \in V' \times \mathbf{Exp}(V') \mid (y, e) \in A_4\}) \ .$$

I.e. we remove all references to x from A_1, A_3 and A_4. We also remove the dimensions of $f \in A_2$ that correspond to x.

The generation proceeds in the opposite direction, being a mapping from $\mathbf{AS}(V')$ to $\mathbf{AS}(V)$. Its definition depends on the arithmetic expression we're analyzing. The generation is performed in the following steps:

1. the determination of variables distributed according to the Laplacian distribution;
2. the computation of values;
3. the computation of averages;
4. the computation of variances;
5. the recording of relationships between the values of variables;
6. the recording of independence relations between variables;
7. the closure of independence relations;
8. the closure of relationships between values;
9. the computation of differential privacy levels.

We now define these steps one after another.

Let $A \in \mathbf{AS}(V')$. Then $gen^1(A)$ returns the set $A'_1 \subseteq V$, for which we know that they are distributed according to the Laplacian distribution.

$$\begin{aligned} gen^1_{x:=x'}(A) &= \begin{cases} A_1, & \text{if} x' \notin A_1 \\ A_1 \cup \{x\}, & \text{if} x' \in A_1 \end{cases} \\ gen^1_{x:=Lap()}(A) &= A_1 \cup \{x\} \\ gen^1_{x:=x_1+x_2}(A) &= \begin{cases} A_1 \cup \{x\}, & \text{if} \exists y_1, y_2 : \{x_1, x_2\} = \{y_1, y_2\} \wedge y_1 \in A_1 \wedge \forall f \in A_2 : f((3, y_2)) = 0 \\ A_1, & \text{otherwise} \end{cases} \\ gen^1_{x:=x_1 \cdot x_2}(A) &= gen^1_{x:=x_1+x_2}(A) \\ gen^1_{x:=\ldots}(A) &= A_1 \end{aligned}$$

Let $f : 3V' + \mathcal{P}((V')^2) \to \mathbb{R}$ be one of possible points in the abstract domain, recording the values of variables and their distribution. Then $gen^2(f)$, $gen^3(f)$ and $gen^4(f)$ give the set of possible values of x after the assignment, the possible set of its average values, and the possible set of its variances in the resulting distribution, respectively. The mappings gen^3 and gen^4 also take the entire abstract state as another argument, allowing them to increase the precision of abstraction.

$$\begin{aligned} gen^2_{x:=x'}(f) &= \{f((1, x'))\} \\ gen^2_{x:=n}(f) &= \{n\} \\ gen^2_{x:=x_1 \otimes x_2}(f) &= \{f((1, x_1)) \otimes f((1, x_2))\}, \text{ where} \otimes \in \{+, \cdot, \leq, =\} \\ gen^2_{x:=Lap()}(f) &= \mathbb{R} \end{aligned}$$

$$
\begin{aligned}
gen^3_{x:=x'}(A,f) &= \{f((2,x'))\} \\
gen^3_{x:=n}(A,f) &= \{n\} \\
gen^3_{x:=x_1+x_2}(A,f) &= \{f((2,x_1)) + f((2,x_2))\} \\
gen^3_{x:=x_1\cdot x_2}(A,f) &= \begin{cases} \{f((2,x_1)) \cdot f((2,x_2))\}, & \text{if} (x_1,x_2) \in A_3 \\ \mathbb{R}, & \text{otherwise} \end{cases} \\
gen^3_{x:=(x_1\, cmp\, x_2)?}(A,f) &= \begin{cases} \{1\}, & \text{if} \forall f' \in A_2 : f'((1,x_1))\ cmp\ f'((1,x_2)) \\ \{0\}, & \text{if} \forall f' \in A_2 : \neg(f'((1,x_1))\ cmp\ f'((1,x_2))) \\ [0,1], & \text{otherwise} \end{cases} \\
gen^3_{x:=Lap()}(A,f) &= \{0\}
\end{aligned}
$$

$$
\begin{aligned}
gen^4_{x:=x'}(A,f) &= \{f((3,x'))\} \\
gen^4_{x:=n}(A,f) &= \{0\} \\
gen^4_{x:=x_1+x_2}(A,f) &= \begin{cases} \{f((3,x_1)) + f((3,x_2))\}, & \text{if} \{\{x_1\},\{x_2\}\} \in A_3 \\ [f((3,x_1)) + f((3,x_2)) - R,\ f((3,x_1)) + f((3,x_2)) + R], & \\ \qquad \text{otherwise, where} R = 2\sqrt{f((3,x_1)) \cdot f((3,x_2))} & \end{cases} \\
gen^4_{x:=x_1\cdot x_2}(A,f) &= \begin{cases} \{f((3,x_1)) \cdot f((2,x_2))^2 + f((3,x_2)) \cdot f((2,x_1))^2 + f((3,x_1)) \cdot f((3,x_2))\}, & \\ \qquad \text{if} \{\{x_1\},\{x_2\}\} \in A_3 & \\ \mathbb{R}, \qquad \text{otherwise} & \end{cases} \\
gen^4_{x:=(x_1\, cmp\, x_2)?}(A,f) &= \begin{cases} \{0\}, & \text{if} \forall f' \in A_2 : f'((1,x_1))\ cmp\ f'((1,x_2)) \\ \{0\}, & \text{if} \forall f' \in A_2 : \neg(f'((1,x_1))\ cmp\ f'((1,x_2))) \\ \mathbb{R}, & \text{otherwise} \end{cases} \\
gen^4_{x:=Lap()}(A,f) &= 1
\end{aligned}
$$

The mapping gen^5 adds the relationships between the newly defined variable x and the others. Here $A_4 \subseteq V' \times \mathbf{Exp}(V')$.

$$
\begin{aligned}
gen^5_{x:=Lap()}(A_4) &= A_4 \\
gen^5_{x:=x'}(A_4) &= A_4 \cup \{(x,x'),(x',x)\} \\
gen^5_{x:=n}(A_4) &= A_4 \cup \{(x,n)\} \\
gen^5_{x:=x_1\otimes x_2}(A_4) &= A_4 \cup \{(x, x_1 \otimes x_2)\}
\end{aligned}
$$

For defining gen^6, adding the independence relations involving x, introduce the following notation. Let $X \subseteq V'$. Let $X_{x_1,\ldots,x_k\to x}$ denote the set $X \cup \{x\}$, if $x_1,\ldots,x_k \in X$, and the set X otherwise. In particular, if $k = 0$, then the introduced notation always denotes $X \cup \{x\}$. Then

$$
\begin{aligned}
gen^6_{x:=Lap()}(A_3) &= \{(X \cup \{x\}, Y) \,|\, (X,Y) \in A_3\} \cup \{(X, Y \cup \{x\}) \,|\, (X,Y) \in A_3\} \\
gen^6_{x:=x_1\otimes\cdots\otimes x_k}(A_3) &= \{(X_{x_1,\ldots,x_k\to x}, Y_{x_1,\ldots,x_k\to x}) \,|\, (X,Y) \in A_3\}
\end{aligned}
$$

Here the last row covers all deterministic operations, including assigning a constant to x.

The closure of the independence relations is found in the following way. Given $A_3 \subseteq (\mathcal{P}(V))^2$, let $A'_3 \subseteq (\mathcal{P}(V))^2$ be the least set satisfying the following conditions:

$$\begin{array}{rcl} & & A_3 \subseteq A'_3 \\ (X,Y) \in A'_3 & \Rightarrow & (Y,X) \in A'_3 \\ (X,Y) \in A'_3, X' \subseteq X, Y' \subseteq Y & \Rightarrow & (X',Y') \in A'_3 \\ (X,Y) \in A'_3, (X \cup Y, Z) \in A'_3 & \Rightarrow & (X, Y \cup Z) \in A'_3 \ . \end{array}$$

Write $A'_3 = gen^7(A_3)$.

The closure of the relationships between values is found in the following way. Given $A_4 \subseteq V \times \mathbf{Exp}(V)$, let $A'_4 \subseteq V \times \mathbf{Exp}(V)$ be the least set satisfying the following conditions:

$$\begin{array}{rcl} & & A_4 \subseteq A'_4 \\ & & \forall x \in V : (x,x) \in A'_4 \\ (x,e) \in A'_4, e \approx e' & \Rightarrow & (x,e') \in A'_4 \\ (x, e(x_1,\ldots,x_k)) \in A'_4, \forall i : (x,e_i) \in A'_4 & \Rightarrow & (x, e(e_1,\ldots,e_k)) \in A'_4 \ . \end{array}$$

In the last row, $x_1, \ldots, x_k$ are all variables occurring in e. Write $A'_4 = gen^8(A_4)$.

Last, we discuss the determination of DP-distances. Let $f, f' : 3V + \mathcal{P}(V^2) \to \mathbb{R}$. We say that $f \leq f'$ if

- $\forall x \in V, i \in \{1,2,3\} : f((i,x)) = f'((i,x))$;
- $\forall Z \subseteq V^2 : f(Z) \leq f'(Z)$.

Let $A = (A_1, A_2, A_3, A_4) \in \mathbf{AS}(V)$ and $f \in A_2$. The *closure* of f in the context of A is the greatest $f' : 3V + \mathcal{P}(V^2) \to \mathbb{R}$ satisfying the following conditions:

$$f' \leq f$$

$$Z' = \{(y,x) \mid (x,y) \in Z\} \Rightarrow f'(Z) \leq f'(Z')$$

$$x,y \in A_1, f'((3,x)) = f'((3,y)) \Rightarrow f'(\{(x,y)\}) \leq |f'((2,x)) - f'((2,y))| / \sqrt{f'((3,x))/2}$$

$$\begin{array}{l} (\{x_1,\ldots,x_s\},\{x'_1,\ldots,x'_t\}),(\{y_1,\ldots,y_s\},\{y'_1,\ldots,y'_t\}) \in A_3 \Rightarrow \\ \quad f'(\{(x_1,y_1),\ldots,(x_s,y_s),(x'_1,y'_1),\ldots,(x'_t,y'_t)\}) \leq f'(\{(x_1,y_1),\ldots,(x_s,y_s)\}) + f'(\{x'_1,y'_1),\ldots,(x'_t,y'_t)\}) \end{array}$$

$$\begin{array}{l} (x, e(x_1,\ldots,x_k)),(x', e(x'_1,\ldots,x'_k)) \in A_4 \Rightarrow \\ f'(\{(x_1,x'_1),\ldots,(x_k,x'_k),(x,x'),(y_1,y'_1),\ldots,(y_\ell,y'_\ell)\}) \leq f'(\{(x_1,x'_1),\ldots,(x_k,x'_k),(y_1,y'_1),\ldots,(y_\ell,y'_\ell)\}) \end{array}$$

Write $f' = gen^9(A,f)$.

The defined pieces $gen^1, \ldots, gen^9$ are combined into the generation function $gen_{x:=e}$. Given $A^\circ \in \mathbf{AS}(V')$, we define $A^\bullet = gen_{x:=e}(A)$ as follows:

1. Let $A^\bullet_1 = gen^1_{x:=e}(A^\circ)$, $A^\bullet_3 = gen^7(gen^6_{x:=e}(A^\circ_3))$ and $A'_4 = gen^8(gen^5_{x:=e}(A^\circ_4))$.
2. For each $f \in A^\circ_2$, let $F_f \in \mathcal{P}(3V + \mathcal{P}(V^2) \to \mathbb{R})$ be the following set:

$$\begin{array}{rl} F_f = \{f' \mid f'|_{3V+\mathcal{P}((V')^2)} = f, f((1,x)) \in gen^2_{x:=e}(f), f((2,x)) \in gen^3_{x:=e}(A^\circ, f), \\ \qquad f((3,x)) \in gen^4_{x:=e}(A^\circ, f), f(Z) = \infty \text{ for all} Z \in V^2 \backslash (V')^2\} \ . \end{array}$$

3. Let $A'_2 = \bigcup_{f \in A^\circ_2} F_f$ and $A' = (A^\bullet_1, A'_2, A^\bullet_3, A^\bullet_4)$.
4. Let $A^\bullet_2 = \{gen^9(A', f) \mid f \in A'_2\}$. Return $A^\bullet = (A^\bullet_1, A^\bullet_2, A^\bullet_3, A^\bullet_4)$.

The abstract semantics of $x := e$ is $gen_{x:=e} \circ kill_x$. We can show that it is a valid abstraction of the concrete semantics.

3.3.3.4 Analysing Branching Programs. Suppose now that our program also contains if-statements. We analyze them by transforming them out again. Indeed, the program

$$\text{if } b \text{ then } s_1 \textit{ else } s_2$$

has the same semantics as the program

$$(x' := x; x'' := x)_{x \in \mathbf{Var}}; s_1'; s_2''; (x := b \ ? \ x' : x'')_{x \in \mathbf{Var}}$$

assuming that no program may diverge. I.e. we execute both branches s_1 and s_2 with renamed variables (renaming x to x' in s_1; and x to x'' in s_2), and afterwards merge them back, using the ternary *choice* operation. To complete the description of the analysis, we only have to provide the transfer function for the statement executing the choice operation.

In order to define the transfer function corresponding to the choice operation, we extend the definitions of gen^i-mappings. We define the first ones as follows:

$$\begin{aligned} gen^1_{x:=b?x':x''}(A) &= A_1 \\ gen^2_{x:=b?x':x''}(f) &= \{f((1,b))?f((1,x')) : f((1,x''))\} \\ gen^3_{x:=b?x':x''}(A,f) &= [\min\{f((2,x')), f((2,x''))\}, \max\{f((2,x')), f((2,x''))\}] \\ gen^4_{x:=b?x':x''} &= \mathbb{R} \end{aligned}$$

The mappings $gen^5, \ldots, gen^8$ stay the same. But we can add to the definition of the closure of $f : 3V + \mathcal{P}(V^2) \to \mathbb{R}$ (in the context of defining gen^9) the following specific rule:

$$\begin{aligned} &(x, b?x_1 : x_2), (x', b'?x_1' : x_2') \in A_4 \Rightarrow \\ &\quad f'(\{(b,b'),(x,x'),(y_1,y_1'),\ldots,(y_\ell,y_\ell')\}) \leq \max\{f'(\{(b,b'),(x_1,x_1'),(y_1,y_1'),\ldots,(y_\ell,y_\ell')\}), \\ &\qquad\qquad f'(\{(b,b'),(x_2,x_2'),(y_1,y_1'),\ldots,(y_\ell,y_\ell')\})\} \end{aligned}$$

With these additions to the mappings gen^i, the definition of the abstract semantics proceeds as in the previous section.

3.3.3.5 Using the Analysis to Characterize a Task. Suppose that a task is specified by the program s, and we want to find its differential privacy level with respect to the input variable x (which the analysis does not change) and the output variable y. We form the program $s; s'$ and analyze it with respect to the initial abstract state A°, where

- $A_1^\circ = \emptyset$, i.e. we do not assume whether any variable is distributed according to a Laplacian distribution in the initial state.
- $A_3^\circ = \emptyset$, i.e. we do not know of any independent sets of variables, either.
- $A_4^\circ = \{(y, y') \mid y \in \mathbf{Var} \backslash \{x\}\}$, meaning that the initial values of the corresponding variables in s and s' are true.
- A_2° similarly expresses the equality between the variables y and y'.

Let $A^\bullet$ be the final state. The differential privacy of the task is conservatively approximated by the mapping $c : \mathbb{R}_+ \to \mathbb{R}_+$, where

$$c(v) = \inf\{f(\{(y,y')\}) \mid f \in A_2^\bullet, |f(x) - f(x')| \geq v\} \ .$$

Depending on the abstraction used for $\mathbf{AS}_2$, this mapping may be represented very succinctly (i.e. $\sup_v c(v)/v$ is explicitly stored in the abstract domain), or it may be invokable as a black box, or it is difficult to access at all. In the latter case, we may need to run the analysis once for each considered distance v of the initial values of x and x', learning the corresponding DP-distance of $D|_y$ and $D|_{y'}$.

3.3.3.6 Language Extensions for SIMD Support. Our programming language has reduced expressivity due to the lack of looping constructs. As next, we consider a construction that gives us some, though not all of the functionality provided by the loops. Nevertheless, the construction is sufficient for expressing many data processors; it is supported by the SecreC language for secure computations [41] and used e.g. in specifying the Rmind privacy-preserving statistics suite [42] running on top of the Sharemind secure multiparty computation platform [43].

We split the set of variables into two: $\mathbf{Var} = \mathbf{SVar} \cup \mathbf{VVar}$, with $\mathbf{SVar}$ be the set of *scalar variables* and $\mathbf{VVar}$ the set of *vector variables*. A scalar variable corresponds to variables in Sec. 3.3.3.1, it holds a value of type $\mathbb{R}$. A vector variable holds a value of type $\mathbb{N} \to \mathbb{R}$ with the restriction that only a finite number of points in this vector are different from 0. In effect, this makes the set of allowed vector values isomorphic to the set of finite sequences of elements of $\mathbb{R}$ not ending with 0.

Fig. 35 gives the extensions to the syntactic categories of our programming language. We introduce two new syntactic categories: $\widetilde{e}$ denotes *arithmetic vector expressions* and $\widetilde{b}$ *boolean vector expressions*. The possible values of the first of them are the same as the values of variables in $\mathbf{VVar}$. The possible values of boolean vector expressions are infinite sequences of booleans (of type $\mathbb{N} \to \mathbb{B}$), where one of the boolean values occurs only finitely many times.

The semantics of the new constructs is given in Fig. 36. The type of the semantics for vector expressions maps a program state to a distribution over vectors of appropriate type. But informally, the meaning of new constructs is the following:

- The vectors of values can be operated pointwise.
- Single elements of the vectors may be selected or assigned.
- vec turns a scalar into a vector, where all elements are equal to that scalar. Actually, for arithmetic vector expressions, vec takes also a second argument that determines, how many first elements of the vector are equal to the scalar. The rest of the elements of the vector are still 0.
- There is an expression to generate random vectors, where each component is randomly distributed according to the Laplacian distribution with mean 0 and variance 1. Again, the expression takes a length argument.
- The vectors may be shifted, i.e. a number of elements (given as the second argument to the shift-operation) at the start of the vectors can be dropped.
- The vectors can be aggregated, delivering a scalar value. We have included in our language some of the most simple aggregation operations. This set may be extended, if necessary.

3.3.3.7 Analysis of SIMD Constructions.

Abstract Domain. We have to modify our abstract domain in order to take into account the unbounded number of locations the program now handles. Previously, the number of the dimensions of the vector space, the subsets of which formed our abstract domain, was finite. We would like to keep it this way.

$$
\begin{array}{rcl}
e & ::= & \ldots \\
 & | & \widetilde{e}[e] \\
 & | & \mathsf{sum}(\widetilde{e})
\end{array}
\qquad
\begin{array}{rcl}
\widetilde{e} & ::= & \widetilde{x} \\
 & | & \widetilde{e}_1 + \widetilde{e}_2 \\
 & | & \widetilde{e}_1 \cdot \widetilde{e}_2 \\
 & | & vLap(e') \\
 & | & \mathsf{shift}(\widetilde{e}, e) \\
 & | & \mathsf{vec}(e_1, e_2)
\end{array}
\qquad
\begin{array}{rcl}
b & ::= & \ldots \\
 & | & \widetilde{b}[e] \\
 & | & \mathsf{any}(\widetilde{b}) \\
 & | & \mathsf{all}(\widetilde{b})
\end{array}
\qquad
\begin{array}{rcl}
\widetilde{b} & ::= & \widetilde{e}_1 \leq \widetilde{e}_2 \\
 & | & \widetilde{e}_1 = \widetilde{e}_2 \\
 & | & \widetilde{b}_1 \wedge \widetilde{b}_2 \\
 & | & \neg \widetilde{b} \\
 & | & \mathsf{shift}(\widetilde{b}, e) \\
 & | & \mathsf{vec}(b)
\end{array}
\qquad
\begin{array}{rcl}
s & ::= & \ldots \\
 & | & \widetilde{x} := \widetilde{e} \\
 & | & \widetilde{x}[e_1] := e_2
\end{array}
$$

Here $\widetilde{x} \in \mathbf{VVar}$.

Figure 35: Syntax of the SIMD Constructions in the Programming Language

$$
\begin{aligned}
[\![\widetilde{e}[e]]\!](S) &= \{\!|\widetilde{v}(i) \mid \widetilde{v} \leftarrow [\![\widetilde{e}]\!](S), i \leftarrow [\![e]\!](S)|\!\} \\
[\![\mathsf{sum}(\widetilde{e})]\!](S) &= \{\!|\sum_{i=0}^{\infty} \widetilde{v}(i) \mid \widetilde{v} \leftarrow [\![\widetilde{e}]\!](S)|\!\}
\end{aligned}
$$

$$
\begin{aligned}
[\![\widetilde{x}]\!](S) &= \eta(S(\widetilde{x})) \\
[\![\widetilde{e}_1 + \widetilde{e}_2]\!](S) &= \{\!|i \mapsto \widetilde{v}_1(i) + \widetilde{v}_2(i) \mid \widetilde{v}_1 \leftarrow [\![\widetilde{e}_1]\!](S), \widetilde{v}_2 \leftarrow [\![\widetilde{e}_2]\!](S)|\!\} \\
[\![\widetilde{e}_1 \cdot \widetilde{e}_2]\!](S) &= \{\!|i \mapsto \widetilde{v}_1(i) + \widetilde{v}_2(i) \mid \widetilde{v}_1 \leftarrow [\![\widetilde{e}_1]\!](S), \widetilde{v}_2 \leftarrow [\![\widetilde{e}_2]\!](S)|\!\} \\
[\![vLap(e')]\!](S) &= \{\!|i \mapsto v'_i \mid \ell \leftarrow [\![e']\!](S), v'_1, \ldots, v'_\ell \leftarrow \mathrm{Lap}^1, v'_{\ell+1} = v'_{\ell+2} = \cdots = 0|\!\} \\
[\![\mathsf{shift}(\widetilde{e}, e)]\!](S) &= \{\!|i \mapsto \widetilde{v}(i+\ell) \mid \widetilde{v} \leftarrow [\![\widetilde{e}]\!](S), \ell \leftarrow [\![e]\!](S)|\!\} \\
[\![\mathsf{vec}(e_1, e_2)]\!](S) &= \{\!|i \mapsto v'_i \mid v \leftarrow [\![e_1]\!](S), \ell \leftarrow [\![e_2]\!](S), v'_1 = \cdots = v'_\ell = v, v'_{\ell+1} = v'_{\ell+2} = \cdots = 0|\!\}
\end{aligned}
$$

$$
\begin{aligned}
[\![\widetilde{b}[e]]\!](S) &= \{\!|\widetilde{v}(i) \mid \widetilde{v} \leftarrow [\![\widetilde{b}]\!](S), i \leftarrow [\![e]\!](S)|\!\} \\
[\![\mathsf{any}(\widetilde{b})]\!](S) &= \{\!|\bigvee_{i=0}^{\infty} \widetilde{v}(i) \mid \widetilde{v} \leftarrow [\![\widetilde{b}]\!](S)|\!\} \\
[\![\mathsf{all}(\widetilde{b})]\!](S) &= \{\!|\bigwedge_{i=0}^{\infty} \widetilde{v}(i) \mid \widetilde{v} \leftarrow [\![\widetilde{b}]\!](S)|\!\}
\end{aligned}
$$

$$
\begin{aligned}
[\![\widetilde{e}_1 \le \widetilde{e}_2]\!](S) &= \{\!|i \mapsto (\widetilde{v}_1(i) \le \widetilde{v}_2(i))? \mid \widetilde{v}_1 \leftarrow [\![\widetilde{e}_1]\!](S), \widetilde{v}_2 \leftarrow [\![\widetilde{e}_2]\!](S)|\!\} \\
[\![\widetilde{e}_1 = \widetilde{e}_2]\!](S) &= \{\!|i \mapsto (\widetilde{v}_1(i) = \widetilde{v}_2(i))? \mid \widetilde{v}_1 \leftarrow [\![\widetilde{e}_1]\!](S), \widetilde{v}_2 \leftarrow [\![\widetilde{e}_2]\!](S)|\!\} \\
[\![\widetilde{b}_1 \wedge \widetilde{b}_2]\!](S) &= \{\!|i \mapsto \widetilde{v}_1(i) \wedge \widetilde{v}_2(i) \mid \widetilde{v}_1 \leftarrow [\![\widetilde{b}_1]\!](S), \widetilde{v}_2 \leftarrow [\![\widetilde{b}_2]\!](S)|\!\} \\
[\![\neg\widetilde{b}]\!](S) &= \{\!|i \mapsto \neg\widetilde{v}(i) \mid \widetilde{v} \leftarrow [\![\widetilde{b}]\!](S)|\!\} \\
[\![\mathsf{shift}(\widetilde{b}, e)]\!](S) &= \{\!|i \mapsto \widetilde{v}(i+\ell) \mid \widetilde{v} \leftarrow [\![\widetilde{b}]\!](S), \ell \leftarrow [\![e]\!](S)|\!\} \\
[\![\mathsf{vec}(b)]\!](S) &= \{\!|i \mapsto v \mid v \leftarrow [\![b]\!](S)|\!\}
\end{aligned}
$$

$$
\begin{aligned}
[\![\widetilde{x} := \widetilde{e}]\!](S) &= \{\!|S[\widetilde{x} \mapsto \widetilde{v}] \mid \widetilde{v} \leftarrow [\![\widetilde{e}]\!](S)|\!\} \\
[\![\widetilde{x}[e_1] := e_2]\!](S) &= \{\!|S[\widetilde{x} \mapsto S(\widetilde{x})[v_1 \mapsto v_2]] \mid v_1 \leftarrow [\![e_1]\!](S), v_2 \leftarrow [\![e_2]\!](S)|\!\}
\end{aligned}
$$

Figure 36: Semantics of SIMD Constructions

We redefine our abstract domain constructor **AS** as follows. For a set of variables V, split into scalar variables V_s and vector variables V_v, we define $\mathbf{AS}(V) = \mathbf{AS}_1(V) \times \mathbf{AS}_2(V) \times \mathbf{AS}_3(V) \times \mathbf{AS}_4(V)$, where the definitions of $\mathbf{AS}_1$, $\mathbf{AS}_3$ and $\mathbf{AS}_4$ stay the same, but

$$\mathbf{AS}_2(V) = \mathcal{P}((3V_\mathrm{s} + 4V_\mathrm{v} + \mathcal{P}(V_\mathrm{s}^2 \cup V_\mathrm{v}^2) \to \mathbb{R}) \times (2V_\mathrm{v} + \mathcal{P}(V_\mathrm{v}^2) \to \mathbb{N}) \times \mathcal{P}(\mathbf{VVar})) \ .$$

I.e. $\mathbf{AS}_2(V)$ gains a number of new dimensions. Comparing it to the previous definition of $\mathbf{AS}_2$, we see that almost all new dimensions are actually integer-valued. We use these dimensions to range over the indices or counts of vector variables, while the existing dimensions range over the values of the elements of vectors pointed out by the integer-values dimensions. Additionally, the subset of vector variables indicates which vectors are constant, i.e. have all elements in their non-zero initial segment equal to each other.

For $D \in \mathcal{D}(\mathbf{State})$, we again define the abstraction $\alpha_2(D) \in \mathbf{AState}_2$. To specify it let us first define the following probability distributions. Let $L_\mathrm{s} = [x_1, \ldots, x_k] \in \mathbf{SVar}^*$ and $L_\mathrm{v} = [(\widetilde{y}_1, I_1), \ldots, (\widetilde{y}_\ell, I_\ell)] \in (\mathbf{VVar} \times \mathbb{N}^*)^*$. For $L = [l_1, \ldots, l_{|L|}] \in \mathbb{N}^*$, and $\widetilde{w} : \mathbb{N} \to \mathbb{R}$ let $\widetilde{w}(L)$ denote the sequence of values

$\widetilde{w}(l_1), \ldots, \widetilde{w}(l_{|L|})$. Define

$$D_{L_s,L_v} = \{\!\{(S(x_1), \ldots, S(x_k), S(\widetilde{y_1})(I_1), \ldots, S(\widetilde{y_\ell})(I_\ell)) \mid S \leftarrow D\}\!\} \ .$$

The set $\alpha_2(D)$ consists of all such pairs (f, g, U) with $f : 3\mathbf{SVar} + 4\mathbf{VVar} + \mathcal{P}(\mathbf{SVar}^2 \cup \mathbf{VVar}^2) \to \mathbb{R}$, $g : 2\mathbf{VVar} + \mathbf{VVar}^2 \to \mathbb{N}$ and $U \subseteq \mathbf{VVar}$, where for some $S \in \underline{D}$:

- $f((1, x)) = S(x)$ for $x \in \mathbf{SVar}$;
- $f((2, x)) = \mathbb{E}[D|_x]$ for $x \in \mathbf{SVar}$;
- $f((3, x)) = \mathbb{V}[D|_x]$ for $x \in \mathbf{SVar}$;
- $g((2, \widetilde{x}))$ is the length of the initial segment of potentially non-zero values in the vector $S(\widetilde{x})$;
- $g((1, \widetilde{x}))$ ranges over $0, \ldots, g((2, \widetilde{x})) - 1$;
- $f((1, \widetilde{x})) = S(\widetilde{x})(g((1, \widetilde{x})))$;
- $f((2, \widetilde{x}))$ is equal to the average of $D|_{x[i]}$, where $i = g((1, \widetilde{x}))$;
- $f((3, \widetilde{x}))$ is equal to the variance of $D|_{x[i]}$, where $i = g((1, \widetilde{x}))$;
- $f((4, \widetilde{x})) = \sum_{i \in \mathbb{N}} S(\widetilde{x})(i)$;
- $\widetilde{x} \in U$, if $S(\widetilde{x})(0) = S(\widetilde{x})(1) = \cdots = S(\widetilde{x})(g((2, \widetilde{x})) - 1)$;
- $g((\widetilde{x}, \widetilde{y})$ ranges over $\mathbb{N}$;
- Let $U = \{(x_1, y_1), \ldots, (x_k, y_k)\} \subseteq \mathbf{SVar}^2$ and $V = \{(\widetilde{x_1}, \widetilde{y_1}), \ldots, (\widetilde{x_\ell}, \widetilde{y_\ell})\} \subseteq \mathbf{VVar}^2$. Then $f(U \cup V)$ is the supermum of all values

  $$d_{\mathrm{dp}}(D_{[x_1,\ldots,x_k],[(\widetilde{x_1},I_1),\ldots,(\widetilde{x_\ell},I_\ell)]}, D_{[y_),\ldots,y_k],[(\widetilde{y_1},I_1),\ldots,(\widetilde{y_\ell},I_\ell)]}),$$

  where $I_1, \ldots, I_\ell \in \mathbb{N}^*$ range over all such sequences of indices, that $|I_r| = g(\widetilde{x_r}, \widetilde{y_r})$ for all $r \in \{1, \ldots, \ell\}$.

Having defined that, the relation $\vDash$ that matches concrete and abstract states can be defined similarly to Sec. 3.3.3.3. We have $D \vDash (A_1, A_2, A_3, A_4)$ if $D \vDash A_i$ for $i \in \{1, 2, 3, 4\}$. These relations are satisfied if

$$\begin{aligned} A_1 \vDash D &\Leftrightarrow \forall x \in A_1 : D|_x \text{ is a Laplace distribution} \\ A_2 \vDash D &\Leftrightarrow \alpha_2(D) \subseteq A_2 \\ A_3 \vDash D &\Leftrightarrow \alpha_3(D) \supseteq A_3 \\ A_4 \vDash D &\Leftrightarrow \alpha_4(D) \supseteq A_4 \ . \end{aligned}$$

If $x \in \mathbf{VVar}$ and $x \in A_1$, then $D|_{x(i)}$ has to be a Laplace distribution for each index i. The mappings α_3 and α_4 are defined as in Sec. 3.3.3.3, this time applying to the entire vectors.

Transfer Functions. The transfer functions for the statements involving SIMD operations are still similar to the ones in Sec. 3.3.3.3. We will thus not give an explicit description of them, but rather discuss them in informal manner. We assume again that each expression in the program contains a single arithmetic / relational operation. There is no change in handling the $\mathbf{AS}_3$- and $\mathbf{AS}_4$-parts of the abstract values: a variable at the left hand side of an assignment is deemed to depend on all variables on the right hand side.

Pointwise operations. The operations $\widetilde{x_1} + \widetilde{x_2}$, $\widetilde{x_1} \cdot \widetilde{x_2}$, etc. are readily supported by our abstraction, introducing constraints like $f((1, \widetilde{x})) = f((1, \widetilde{x_1})) + f((1, \widetilde{x_2}))$ into an element $(f, g, U) \in A_2$, if $g((1, \widetilde{x})) = g((1, \widetilde{x_1})) = g((1, \widetilde{x_2}))$. If the values of $g((1, \cdot))$ are different for $\widetilde{x}$, $\widetilde{x_1}$ and/or $\widetilde{x_2}$, then we obtain no constraints here, unless the study of other elements of (f, g, U) gives us any further relationships that always hold.

For pointwise addition, we can also put $f((4, \widetilde{x})) = f((4, \widetilde{x_1})) + f((4, \widetilde{x_2}))$, i.e. the sum of all elements of $\widetilde{x}$ is equal to the sum of the sums of all elements $\widetilde{x_1}$ and $\widetilde{x_2}$. For multiplication, we use the constancy indicator in U to decide whether we get any constraint on $f((4, \widetilde{x}))$. The constancy indicators of $\widetilde{x_1}$ and $\widetilde{x_2}$ are used to decide on the constancy of $\widetilde{x}$.

Array element assignment. The operation $\widetilde{x}[x_1] := x_2$ is interpreted as an arithmetic operation, where the result is stored in $\widetilde{x}$, and where the operands are $\widetilde{x}$ (before killing it), x_1 and x_2. We keep the same constraints for $f((1,\widetilde{x}))$, $f((2,\widetilde{x}))$ and $f((3,\widetilde{x}))$ as before, as long as $g((1,\widetilde{x})) \neq f((1,x_1))$. If $g((1,\widetilde{x})) = f((1,x_1))$, then $f((1,\widetilde{x})) = f((1,x_2))$. We also get the equalities $f((2,\widetilde{x})) = f((2,x_2))$ and $f((3,\widetilde{x})) = f((3,x_2))$, if $g((1,\widetilde{x})) = f((1,x_1))$ and $f((3,x_1)) = 0$, i.e. the index x_1 is non-probabilistic.

If $g((1,\widetilde{x})) = f((1,x_1))$, then we can also update the sum of the vector $\widetilde{x}$: $f((4,\widetilde{x}))$ is added the difference of current and previous $f((1,\widetilde{x}))$. Otherwise, there are no constraints for $f((4,\widetilde{x}))$.

Shifts. The assignment $\widetilde{x_1} = \mathsf{shift}(\widetilde{x_2}, x_3)$ introduces equalities between $f((1,\widetilde{x_1}))$ and $f((1,\widetilde{x_2}))$, if $g((1,\widetilde{x_1})) + f((1,x_3)) = g((1,\widetilde{x_2}))$.

Promotions to vector. The assignment $\widetilde{x} = \mathsf{vec}(x_1, x_2)$ introduces equalities between $f((1,\widetilde{x}))$ and $f((1,x_1))$, as long as $f((1,x_2)) > g((1,\widetilde{x}))$. Equalities for the second and third component follow if x_2 is constant (has 0 variance).

Summations. The fourth component — $f((4,\widetilde{x}))$ — has been introduced specifically to handle the sums.

Differential privacy. Our operations are such that the DP-distance between the projections of the concrete distribution D involving n elements of some vector $\widetilde{x}$ is typically n times larger than the distance between projections involving just a single element of $\widetilde{x}$. We can compute these distances similarly to gen^9 in the previous paragraph. In operations involving the summation of the vector $\widetilde{x}$, the value $g((2,\widetilde{x}))$ is consulted to find out on how many elements of $\widetilde{x}$ the sum depends on.

3.3.3.8 Abstracting the Set of Points. Our abstract domain is too large to be directly implemented, especially $\mathbf{AState}_2$, the elements of which are sets of points in a many-dimensional vector space. By the definition of $\vDash$, it is safe to add extra points to these sets. Hence we may in our implementation consider not all subsets of this vector space, but only certain ones, such that for each subset there is a considered set that contains it.

A *numeric lattice* is a tractable abstraction of the set of subsets of $\mathbb{R}^n$ (or $\mathbb{N}^n$). Each element a of the lattice corresponds to a certain subset $\gamma(a) \subseteq \mathbb{R}^n$, and a concrete subset R is safely abstracted by any lattice element a for which $R \subseteq \gamma(a)$. A numeric lattice comes with operations to drop a dimension (i.e. to kill a variable) and to add a new dimension, with the new coordinate being somehow constrained by the existing coordinates of the points (i.e. to generate a new value of a variable; the constraints that are easily handled vary among different lattice constructions).

Numeric lattices are used in program analyses to keep track of the possible values that the variables may take in the current program point. We are going to use them for abstracting $\mathbf{AS}_2(V)$ for sets of variables V.

A number of different constructions of numeric lattices for $\mathbb{R}^n$ and $\mathbb{N}^n$ have been proposed. For us, the useful ones may be arithmetic intervals [39], linear equalities [44] and inequalities [45], perhaps also linear congruences [46]. One may also combine different lattices, and the combination of intervals and linear equalities is often found to be both sufficiently precise and efficient. When implementing our analysis, we intend to start with this lattice and add more precision when necessary.

3.3.3.9 Abstracting Other Components of the Abstract State. The constructors $\mathbf{AS}_1$, $\mathbf{AS}_3$ and $\mathbf{AS}_4$ give us much more tractable abstract domains and we believe that in order to represent their elements, no further abstraction is necessary. Indeed,

- we have $\mathbf{AS}_1(V) = \mathcal{P}(V)$. The elements of $\mathcal{P}(V)$ are simply subsets of V, where V is a rather small, finite set. Any representation of sets will do for elements of $\mathbf{AS}_1(V)$.

- we have $\mathbf{AS}_3(V) = \mathcal{P}((\mathcal{P}(V))^2)$, with certain closure properties given by the mapping gen^7. We have had previous success [47, Chap. 5] in representing the elements of $\mathbf{AS}_3(V)$ as binary decision diagrams (BDD), using the isomorphism $\mathcal{P}((\mathcal{P}(V))^2) \cong \mathbb{B}^{2V} \to \mathbb{B}$. Our experience shows that thanks to the monotonicity properties, the BDDs do not grow too large and can be efficiently stored in memory.
- we have $\mathbf{AS}_4(V) = \mathcal{P}(V \times \mathbf{Exp}(V))$. The set $\mathbf{Exp}(V)$ is in general infinite. But according to the definition of $\models$, it is safe to drop pairs (x, e) from the abstraction of a distribution D. We thus expect to store a finite, but useful subset of the actual fourth component of the abstract state.

3.3.4 Mutual Information in Workflows. We have proposed a method that takes the descriptions of tasks in the form of the amount of *Shannon's mutual information* between its inputs and outputs; and returns the same for the entire workflow.

Our analysis works with the notions related to Shannon's entropy. Different quantitative measures of information flow may be most relevant in different situations, and Shannon's entropy is often not the best choice, particularly when the inputs to a system are distributed in a highly skewed manner. Still, these notions enjoy some simple identities that make the analysis tractable.

3.3.4.1 Preliminaries of Information Theory. A *random variable* X takes values in some set $\mathbf{X}$ with certain probabilities. For an element $x \in \mathbf{X}$ we let $p_X(x)$ denote the probability that the value of X is x. If X and Y are two random variables over sets $\mathbf{X}$ and $\mathbf{Y}$, respectively, then "X, Y" is also a random variable taking values in the set $\mathbf{X} \times \mathbf{Y}$. Two different random variables do not have to be *independent* — the quantities $p_{X,Y}(x, y)$ and $p_X(x) \cdot p_Y(y)$ may differ. Here $p_{X,Y}(x, y)$ denotes the probability $\Pr[X = x, Y = y]$. We similarly introduce the notation $p_{X,Y}(x|y)$ for the conditional probability $\Pr[X = x|Y = y]$.

In the mutual information analysis we only consider *discrete* random variables, i.e. such variables X, where $p_X(x) \neq 0$ only for finitely many different x. For such random variable, its *(Shannon) entropy* is defined by

$$H(X) = -\sum_{x \in \mathbf{X}} p_X(x) \log p_X(x),$$

where the logarithm is in base 2, and where we define $0 \cdot \log 0$ to be 0. The entropy of a random variable is a possible measure for the amount of information conveyed through its value. The entropy of X may be conditioned over another variable Y taking a particular value y, giving

$$H(X|Y = y) = -\sum_{x \in \mathbf{X}} p_{X,Y}(x|y) \log p_{X,Y}(x|y)$$
$$H(X|Y) = \sum_{y \in \mathbf{Y}} p_Y(y) \cdot H(X|Y = y),$$

where the latter is the *conditional entropy* of X given Y, describing how much extra information X gives if we already know Y.

The conditional entropy and the (joint) entropy are related by the equality $H(X, Y) = H(X) + H(Y|X)$. Symmetrically, $H(X, Y) = H(Y) + H(X|Y)$. Hence $H(X) - H(X|Y) = H(Y) - H(Y|X)$. This quantity is called the *mutual information* of X and Y and denoted $I(X; Y)$. It can also be conditioned over another random variable: $I(X; Y|Z) = H(X|Z) - H(X|Y, Z)$.

Mutual information $I(X; Y)$ characterizes the mutual dependence between two random variables X and Y. If X characterizes the distribution of inputs to some process, and Y the corresponding distribution of outputs, then $I(X; Y)$ is the amount of information that flows through that process, from X to Y.

Fact. If X, Y, Z, W are random variables, then $I(X; Y|Z, W) \leq I(X, W; Y|Z)$. This follows easily from the relationships between mutual information and (conditional) entropy [48, Sec. 2.4].

3.3.4.2 The Workflows that we Consider. A *workflow* consists of information processing *components*, composed sequentially and/or in parallel. The components are connected by *wires*.

Let *Ports* be a fixed infinite set, the elements of which are called *ports*. For each $p \in Ports$ let $\mathrm{V}(p)$ be the set of values that can be input or output through port p. For a set X, et $\mathcal{D}(X)$ denote the set of probability distributions over X.

Definition 3.6. A *component* is a tuple $M = (\mathsf{ip}_M, \mathsf{op}_M, f_M)$, where $\mathsf{ip}_M, \mathsf{op}_M \subset Ports$ are finite, $\mathsf{ip}_M \cap \mathsf{op}_M = \emptyset$, and $f_M : \prod_{p \in \mathsf{ip}_M} \mathrm{V}(p) \to \mathcal{D}(\prod_{p \in \mathsf{op}_M} \mathrm{V}(p))$.

Definition 3.7. A *workflow* is a tuple $WF = (\mathcal{M}, \mathcal{W}, \mathsf{s}, \mathsf{t})$, where $\mathcal{M}$ is a finite set of components, $\mathcal{W}$ is the finite set of *wires*, $\mathsf{s} : \sum_{M \in \mathcal{M}} \mathsf{op}_M \to \mathcal{W}$ and $\mathsf{t} : \sum_{M \in \mathcal{M}} \mathsf{ip}_M \to \mathcal{W}$, satisfying the following constraints:

- The mapping s is injective.
- For any two ports p_1, p_2 of the components of WF, if $\mathsf{s}(p_1) = \mathsf{t}(p_2) = w$ or $\mathsf{t}(p_1) = \mathsf{t}(p_2) = w$, then $\mathrm{V}(p_1) = \mathrm{V}(p_2)$. We denote this set by $\mathrm{V}(w)$.
- There are no cycles in the directed graph having the tasks in $\mathcal{M}$ as vertices, where an arc from M_1 to M_2 exists iff there exist $p_1 \in \mathsf{op}_{M_1}$ and $p_2 \in \mathsf{ip}_{M_2}$, such that $\mathsf{s}(p_1) = \mathsf{t}(p_2)$.

We introduce the following workflow-related notions:

- The *inputs* or *input wires* of a component M are the wires in the set $\mathcal{I}_M = \mathsf{t}(\mathsf{ip}_M)$. Similarly, the *outputs* of M are the wires in $\mathcal{O}_M = \mathsf{s}(\mathsf{op}_M)$.
- The *listeners* of a wire w are the components M satisfying $w \in \mathcal{I}_M$.
- A *path* in the workflow is an alternating list of wires and components, each wire followed by one of its listeners and each component by one of its output wires.
- A wire w is a global input of the workflow, if $\mathsf{s}(p) \neq w$ for all output ports of all components in the workflow. Denote the set of all global inputs by $\mathcal{G}$.

Definition 3.8. Let $WF = (\mathcal{M}, \mathcal{W}, \mathsf{s}, \mathsf{t})$ be a workflow with global inputs $\mathcal{G}$. Let $InpDist \in \mathcal{D}(\mathcal{G} \to \mathbf{V})$. The *run* of WF starting from $InpDist$ is a random variable of type $\mathcal{W} \to \mathbf{V}$, sampled as follows:

- The values for all $w \in \mathcal{G}$ are sampled from the distribution $InpDist$;
- Each component $\mathcal{M}$ for which all of its input wires are already mapped to values, applies f_M to the tuple of values at its input ports, probabilistically producing a tuple of values for its output ports. These values are added to the mapping for the output wires of M.
- The previous item is repeated until all wires are mapped (this terminates because there are no cycles in the workflow).

We identify wires and their corresponding random variables. Also each set of wires is identified with a tuple of random variables (in some order of wires, fixed for the workflow) considered as a single composite random variable. Thus we can write $I(\mathcal{A}; C)$ as the mutual information between the sets of wires $\mathcal{A}$ and C.

Lemma 3.5. *Let $\mathcal{A}$ be the set of all input wires of a component M. Let $\mathcal{B}$ be subset of the output wires of M. Let C be a subset of wires into which there is no path from M. Then $I(\mathcal{B}; C|\mathcal{A}) = 0$.*

Proof. Follows from the definition of the run. □

For each set of wires $\mathcal{X}$, let

- $\mathrm{V}(\mathcal{X})$ be the set of possible values of the wires $\mathcal{X}$,
- $\mathrm{d}(\mathcal{X})$ be the distribution of the values on the wires $\mathcal{X}$,
- $\mathcal{D}(\mathcal{X})$ be the set of all distributions over $\mathrm{V}(\mathcal{X})$,
- $\mathrm{Const}(\mathcal{X})$ be the set of all constant distributions (also called degenerate distributions or deterministic distributions) over $\mathrm{V}(\mathcal{X})$.

For each value v, let Const(v) be the constant distribution of the value v.

As inputs to the analysis, each component may have a description about the known bounds on the information flow from some subsets of its inputs to some subsets of its outputs. If there are no known bounds then the flows can be infinite. Each wire may also have a bound on the size of the values sent over that wire.

We are interested in $I(S;T)$. This quantity is uniquely determined by the distribution d(G) of the values of the global inputs G and the conditional distributions $P_{O_M|\mathcal{I}_M=\vec{a}}$ for all M and $\vec{a}$ (which together induce the distribution of the values of all wires). We do not necessarily know these distributions exactly. Instead, the input of our analyzer includes declarations that restrict the distribution d(G) to a subset of $\mathcal{D}(G)$ and for all M, restrict the function f where $f(\vec{a}) = P_{O_M|\mathcal{I}_M=\vec{a}}$, to a subset of $\mathcal{D}(O_M)^{\mathrm{V}(\mathcal{I}_M)}$.

S may be a proper subset of $\mathcal{G}$. In this case, the global inputs $\mathcal{G} \setminus S$ are not considered sensitive. Thus, we may assume that the adversary already knows the value of $\mathcal{G} \setminus S$, i.e. its distribution may be considered constant. Thus d($\mathcal{G} \setminus S$) $\in$ Const($\mathcal{G} \setminus S$). Therefore, our goal is to compute an upper bound on the value

$$\max_{\substack{\mathrm{d}(S)\in\mathcal{D}_S \\ \mathrm{d}(\mathcal{G}\setminus S)\in\mathrm{Const}(\mathcal{G}\setminus S)}} I(S;T) \tag{6}$$

where $\mathcal{D}_S$ is the set of distributions into which the distribution of S is known to belong, according to the sensitivity declarations that will be described in Sec. 3.3.4.4.

We compute an upper bound on $I(S;T)$ using similar bounds for the individual components, i.e. $I(\mathcal{A},C)$ for each component M, for all $\mathcal{A} \subseteq \mathcal{I}_M, C \subseteq O_M$.

Let $\mathcal{I} = \mathcal{I}_M$ and $C \subseteq O_M$. For each $\mathcal{A} \subseteq \mathcal{I}$ and $\mathcal{D} \subseteq \mathcal{D}(\mathcal{A})$, let

$$q_M^{\mathcal{D}}(\mathcal{A};C) = \max_{\substack{\mathrm{d}(\mathcal{A})\in\mathcal{D} \\ \mathrm{d}(\mathcal{I}\setminus\mathcal{A})\in\mathrm{Const}(\mathcal{I}\setminus\mathcal{A})}} I(\mathcal{A};C) \tag{7}$$

We take the maximum over all distributions that $\mathcal{A}$ may belong to because we do not know what the actual distribution on $\mathcal{A}$ is and we want $q_M^{\mathcal{D}}(\mathcal{A};C)$ to be an upper bound on $I(\mathcal{A};C)$. If we do not have any knowledge about the distribution of $\mathcal{A}$ then $\mathcal{D} = \mathcal{D}(A)$. If we have already determined the possible distributions of the inputs $\mathcal{I}_M$ (as will be described in Sec. 3.3.4.4) then we can write $q_M(\mathcal{A};C)$ instead of $q_M^{\mathcal{D}}(\mathcal{A};C)$.

The description of a component M should ideally contain the values $q_M(\mathcal{A};C)$ for all subsets of $\mathcal{A}$ of the inputs of M and all subsets C of the outputs of M. Because it may be difficult to determine the values $q_M(\mathcal{A};C)$, we may instead have upper bounds on these values. Also, we may not have the values for all $\mathcal{A}$ and C.

The triangle equality does not hold for q_M. Thus it is possible that

$$q_M(\mathcal{A}_1;C) + q_M(\mathcal{A}_2;C) < q_M(\mathcal{A}_1 \cup \mathcal{A}_2;C)$$

or

$$q_M(\mathcal{A};C_1) + q_M(\mathcal{A};C_2) < q_M(\mathcal{A};C_1 \cup C_2)$$

Thus it does not in general suffice to give $q_M(\mathcal{A};C)$ only for one-element sets $\mathcal{A}$ and C, because no bounds for larger sets of wires can be deduced from these.

Monotonicity does hold:

$$\mathcal{A}' \subseteq \mathcal{A} \wedge C' \subseteq C \Rightarrow q_M(\mathcal{A}';C') \leq q_M(\mathcal{A};C)$$

but it may not give the best upper bound on $q_M(\mathcal{A}';C')$.

3.3.4.3 Differential Privacy. Because q_M does not satisfy the triangle inequality, we may instead use a different quantity that does satisfy the triangle inequality and that implies a bound on q_M.

Definition 3.9. Let P be a probability distribution over $\mathcal{A}$. Denote by $d_M^P(\mathcal{A};C)$ the least value ε (which may also be ∞) such that for all value tuples $\vec{a}$ and $\vec{a}'$ of the inputs $\mathcal{A}$ for which $P(\vec{a}) > 0$ and $P(\vec{a}') > 0$, for all value tuples $\vec{b}$ of the inputs $\mathcal{I}_M \setminus \mathcal{A}$,

$$d_{\mathrm{dp}}(P_{C|\mathcal{A}=\vec{a},\mathcal{I}_M\setminus\mathcal{A}=\vec{b}}, P_{C|\mathcal{A}=\vec{a}',\mathcal{I}_M\setminus\mathcal{A}=\vec{b}}) \le \varepsilon \ .$$

For any set $\mathcal{D}$ of probability distributions over $\mathcal{A}$, let

$$d_M^{\mathcal{D}}(\mathcal{A};C) = \max_{P\in\mathcal{D}} d_M^P(\mathcal{A};C)$$

We have the following connection between differential privacy and information flow.

Lemma 3.6. *Let $d_M^{\mathcal{D}}(\mathcal{A};C) = \varepsilon$. Then $q_M^{\mathcal{D}}(\mathcal{A};C) \le q$ bits, where*

$$q = \varepsilon \frac{(e^\varepsilon - 1)(1 - e^{-\varepsilon})}{(e^\varepsilon - 1) + (1 - e^{-\varepsilon})} \cdot \frac{1}{\ln 2} \tag{8}$$

Proof. The proof is similar to [49]. Let $D(P \parallel Q) = \sum_{\mathbf{a}} P(\mathbf{a}) \log(P(\mathbf{a})/Q(\mathbf{a}))$ be the Kullback-Leibler divergence from Q to P.

We have $\forall P \in \mathcal{D} : \ P(\vec{a}) > 0 \wedge P(\vec{a}') > 0. \ \forall \vec{b}. \ P_{C|\mathcal{A}=\vec{a},\mathcal{I}_M\setminus\mathcal{A}=\vec{b}}$ and $P_{C|\mathcal{A}=\vec{a}',\mathcal{I}_M\setminus\mathcal{A}=\vec{b}}$ are ε-close. This is analogous to the statement [49] that for all neighboring databases x^n and $\tilde{x}^n$, $P_{Y|X^n=x^n}$ and $P_{Y|X^n=\tilde{x}^n}$ are ε-close. Both of these statements characterize ε-differential privacy.

Cuff and Yu [49] show that if P and Q are ε-close, then $D(P \parallel Q) \le q$ bits and $D(Q \parallel P) \le q$ bits where q is as in (8). I.e. we have $\forall P \in \mathcal{D} : \ P(\vec{a}) > 0 \wedge P(\vec{a}') > 0. \ \forall \vec{b}. \ D(P_{C|\mathcal{A}=\vec{a},\mathcal{I}_M\setminus\mathcal{A}=\vec{b}} \parallel P_{C|\mathcal{A}=\vec{a}',\mathcal{I}_M\setminus\mathcal{A}=\vec{b}}) \le q$ bits.

Consider any case where $P \in \mathcal{D}$, $\mathcal{A} \sim P$, and $\mathcal{I}_M \setminus \mathcal{A} = \vec{b}$. Then, analogously to [49],

$$\begin{aligned}
&I(\mathcal{A};C) = \mathbb{E}_{\mathcal{A}} D(P(C|\mathcal{A}) \parallel P(C)) = \\
&= \mathbb{E}_{\mathcal{A}} D(P_{C|\mathcal{A}=\vec{a},\mathcal{I}_M\setminus\mathcal{A}=\vec{b}} \parallel \mathbb{E}_{P(\vec{a}')>0} P_{C|\mathcal{A}=\vec{a}',\mathcal{I}_M\setminus\mathcal{A}=\vec{b}}) \le \\
&\le \mathbb{E}_{\mathcal{A}} \mathbb{E}_{P(\vec{a}')>0} D(P_{C|\mathcal{A}=\vec{a},\mathcal{I}_M\setminus\mathcal{A}=\vec{b}} \parallel P_{C|\mathcal{A}=\vec{a}',\mathcal{I}_M\setminus\mathcal{A}=\vec{b}}) \le q \text{ bits}
\end{aligned}$$

Because this holds for all considered cases, we have $q_M^{\mathcal{D}}(\mathcal{A};C) \le q$ bits. □

If we have already determined the possible distributions of the inputs $\mathcal{I}_M$ (as will be described in Sec. 3.3.4.4) then we can write $d_M(\mathcal{A};C)$ instead of $d_M^{\mathcal{D}}(\mathcal{A};C)$. Then d_M satisfies triangle inequality for inputs:

$$d_M(\mathcal{A}_1;C) + d_M(\mathcal{A}_2;C) \ge d_M(\mathcal{A}_1 \cup \mathcal{A}_2;C)$$

Thus the description of a component may give $d_M(\mathcal{A};C)$ only for the cases where $\mathcal{A}$ is a one-element set, then we can use the triangle inequality to find an upper bound on $d_M(\mathcal{A};C)$ for the cases where $\mathcal{A}$ is a larger set, and then convert this to an upper bound on $q_M(\mathcal{A};C)$.

Note that d_M may not satisfy triangle inequality for outputs. If the outputs C_1 and C_2 are calculated from the input A (which is in some bounded range) by adding r and $-r$ to them, respectively, where r is a Laplace random value, then $d_M(A;C_1) = d_M(A;C_2)$ is finite but $d_M(A;C_1,C_2) = \infty$ because the randomness in C_1 and C_2 can be canceled out, revealing the exact value of A.

Differential privacy is useful for bounding leakages of information from a certain provenance but it may not always give the best bounds. For example, if we make in parallel 100 queries, each 0.1-differentially private, then the combination is 10-differentially private. When converted to mutual information (using (8)), this gives 14.4 bits of leakage. On the other hand, each 0.1-differentially private query separately, when converted to mutual information, leaks 0.0072 bits. Because results of the queries are conditionally independent (conditioned on the inputs), the triangle inequality holds here for mutual information, thus the 100 queries together leak only 0.72 bits, not 14.4 bits. Thus we get a much better bound on the leakage. This gives motivation for combining differential privacy and mutual information when bounding leakages.

Note that, in (8), $q \approx \frac{\varepsilon^2}{2 \ln 2}$ when ε is small. This is one of the reasons that we use Shannon entropy instead of min-entropy. If we used min-entropy, we would get the bound $q = \frac{\varepsilon}{\ln 2}$ [50], even when ε is small. When the output C can have only 2 possible values then [50] gives an improved bound $q \approx \frac{\varepsilon}{2 \ln 2}$ when ε is small. Now consider the example in the previous paragraph. Each 0.1-differentially private query, when converted to min-entropy, leaks at most 0.0703 bits of min-entropy if the output is binary, and 0.144 bits in the general case. The 100 queries together leak either 7.03 or 14.4 bits. Thus, combining differential privacy with min-entropy during the whole analysis, we would get no or only a small improvement over the bound (14.4 bits) that we get when using only differential privacy in the analysis and converting the final result to min-entropy. On the other hand, as described in the previous paragraph, combining differential privacy with Shannon entropy during the whole analysis improves the bound 20 times compared to using only differential privacy in the analysis and converting the final result to mutual information.

3.3.4.4 Inputs to the Analysis. Our information-flow analysis takes as input the graphical description of the workflow — the names of tasks and ports, as well as the wires from one port to another. It takes as input the subsets S and T of wires, stating which global inputs contain sensitive information, and which wires are read by the adversary. It also takes as input the information flow behaviour of tasks. The latter may be expressed in many different kinds, which we describe below.

Sensitivity. For each wire w, let dist_w be a distance (metric) on $\mathrm{V}(w)$. Let

$$\beta_0(w) = \max_{a,a' \in \operatorname{supp} \mathrm{d}(w)} \mathrm{dist}_w(a, a') \tag{9}$$

This is the diameter (according to dist_w) of the support of the distribution of w.

Our analysis can make use of declarations that the support of the distribution of a global input w has diameter (according to dist_w) at most s. In this case, let $\beta(w) = s$. For those global inputs w for which there is no such declaration, let $\beta(w) = \infty$. Then $\beta(w) \geq \beta_0(w)$ for all global inputs w.

Our analysis can also make use of declarations that (M, A, C) (where $A \in \mathcal{I}_M, C \in O_M$) has c-sensitivity. This means that

- for all $a, a' \in \mathrm{V}(A), \mathbf{b} \in \mathrm{V}(\mathcal{I}_M \setminus \{A\}), d, d' \in \mathrm{V}(C)$:
  - if M may output d on C if it gets a on A and $\mathbf{b}$ on $\mathcal{I}_M \setminus \{A\}$
  - and M may output d' on C when it gets a' on A and $\mathbf{b}$ on $\mathcal{I}_M \setminus \{A\}$
  - then $\mathrm{dist}_C(d, d') \leq c \cdot \mathrm{dist}_A(a, a')$.

In other words, if we change the input A by a certain distance then the output C can change by at most c times that distance. The component M may have sensitivity declarations for several pairs of its inputs and outputs. Denote $c(A, C) = c$ if (M, A, C) has c-sensitivity and $c(A, C) = \infty$ if there does not exist c such that (M, A, C) has c-sensitivity, or such c has not been given.

All sensitivity declarations involving a certain wire (either as an input or an output of a component, or as a global input) must use the same distance dist_w on the values of that wire. If the values are databases then distance may be e.g. the number of records differing in the two versions of the database. If the values are scalars then the distance may be the absolute value of the difference of the two versions of the value.

If we know $\beta(A)$ and dist_A for all $A \in \mathcal{A}$ then we can find the set of distributions $\mathcal{D}$ used implicitly in $d_M(\mathcal{A}; C)$ and $q_M(\mathcal{A}; C)$ to denote $d_M^{\mathcal{D}}(\mathcal{A}; C)$ and $q_M^{\mathcal{D}}(\mathcal{A}; C)$, respectively:

$$\mathcal{D} = \mathcal{D}_{\mathcal{A}} = \{P \mid \forall A \in \mathcal{A}, a, a' \in \operatorname{supp} P|_A.\ \mathrm{dist}_A(a, a') \leq \beta(A)\} \tag{10}$$

Differential privacy. Consider a component M and one of its inputs A. Let d_{dp} be the differential-privacy distance defined on the distributions of a subset of its outputs C.

Our analysis can make use of declarations that (M, A, C) has ε-differential privacy. This means that for all $a, a', \vec{b}$:

$d_{\mathrm{dp}}(P_{C|A=a,\mathcal{I}_M\setminus\{A\}=\vec{b}}, P_{C|A=a',\mathcal{I}_M\setminus\{A\}=\vec{b}}) \le \varepsilon \cdot \mathrm{dist}_A(a,a')$. If such declaration exists for some M, A, and C, then denote this value ε by $\varepsilon(A,C)$. Put $\varepsilon(A,C)=\infty$, if no such declaration exists.

Our analysis can also make use of declarations that $(M,\mathcal{A},C)$ has sensitivity-less ϵ-differential privacy. This means that for all $\vec{a},\vec{a'},\vec{b}$:
$d_{\mathrm{dp}}(P_{C|\mathcal{A}=\vec{a},\mathcal{I}_M\setminus\mathcal{A}=\vec{b}}, P_{C|\mathcal{A}=\vec{a'},\mathcal{I}_M\setminus\mathcal{A}=\vec{b}}) \le \epsilon$. If such declaration exists for some M, A, and C, then denote this value ϵ by $\epsilon(A,C)$. Put $\epsilon(A,C)=\infty$, if no such declaration exists.

Mutual information. Our analysis can make use of declarations that a component M leaks at most q bits from a subset $\mathcal{A}_i$ of its inputs to a subset C_j of its outputs, i.e. $q_M^{\mathcal{D}(\mathcal{A}_i)}(\mathcal{A}_i,C_j) \le q$. This implies $q_M(\mathcal{A}_i,C_j)\le q$. These are the mutual information declarations for $(M,\mathcal{A}_i,C_j)$, meaning that $(M,\mathcal{A}_i,C_j)$ has at most q bits of mutual information. Here the triangle inequality does not hold.

3.3.4.5 Analysis. The goal of our analysis is to conservatively estimate (i.e. upper-bound) (6). To compute it, we make several passes over the description of the workflow. These passes result us in finding $q_M(\mathcal{A},C)$ for each component M, for all subsets $\mathcal{A}$ of its inputs and all subsets C of its outputs. We will then invoke a graph-theoretic algorithm that computes (6) from all $q_M(\mathcal{A},C)$. We describe the computations below.

Computing β for all wires. In Sec. 3.3.4.4, we defined $\beta(w)$ for all global inputs w and we showed that it is an upper bound of $\beta_0(w)$ (9) in this case. For any other wire C (taken in topological order), which belongs to O_M for some component M, we can compute $\beta(C)$ as

$$\beta(C) = \sum_{A\in\mathcal{I}_M} \beta(A)\cdot c(A,C)$$

It is easy to see, by induction and using the triangle inequality for dist_C, that $\beta(w)\ge\beta_0(w)$ for all wires w. If we know that $\beta(w)=s$ then we know that the distribution of the values on w is such that any two values with non-zero probability are at a distance at most s from each other.

Parallel composition of differential privacy. For each component M and $\mathcal{A}\subseteq\mathcal{I}_M, C\subseteq O_M$, let

$$\gamma(M,\mathcal{A},C) = \min\{\epsilon(\mathcal{A},C), \sum_{A\in\mathcal{A}} \min\{\varepsilon(A,C)\cdot\beta(A), \epsilon(A,C)\}\} \ .$$

It is easy to see that $\epsilon(\mathcal{A},C) \ge d_M^{\mathcal{D}(\mathcal{A})}(\mathcal{A},C)$, $\epsilon(A,C)\ge d_M^{\mathcal{D}(A)}(A,C)$, $\varepsilon(A,C)\cdot\beta(A)\ge d_M(A,C)$. Now, using the triangle inequality for d_{dp}, we get that

$$\gamma(M,\mathcal{A},C) \ge d_M(\mathcal{A},C) = d_M^{\mathcal{D}}(\mathcal{A},C), \tag{11}$$

where $\mathcal{D}$ is as in (10).

Bounding the mutual information through a component. Consider a component M. Let $\mathcal{A}$ be the subset of its inputs and C the subset of its outputs that are on the path from the source to the sink. Suppose we want to find a bound on how much information can flow through M from $\mathcal{A}$ to C, i.e. an upper bound on $q_M^{\mathcal{D}}(\mathcal{A};C)$, where $\mathcal{D}$ is the set of distributions into which the actual distribution of $\mathcal{A}$ is known to belong. $\mathcal{D}$ is determined by the sensitivity declarations, as described in Sec. 3.3.4.4. If there are no sensitivity declarations about the wires in $\mathcal{A}$ then $\mathcal{D}=\mathcal{D}(A)$.

If we have a mutual-information declaration for $(M,\mathcal{A},C)$ then we can use the bound from that declaration. If we have a mutual-information declaration for $(M,\mathcal{A}',C')$ where $\mathcal{A}\subseteq\mathcal{A}'$ and $C\subseteq C'$ then by monotonicity we can also use that bound. If we get bounds from several declarations then we take the minimum of those bounds.

If we have a differential-privacy declaration for $(M,\mathcal{A},C)$ then we use that to find an upper bound on $d_M(\mathcal{A},C)$. If we have differential-privacy declarations for (M,A,C) for each $A\in\mathcal{A}$ then we use (11) to find an upper bound on $d_M(\mathcal{A},C)$. Then we convert the bound on $d_M^{\mathcal{D}}(\mathcal{A},C)$ to a bound on $q_M^{\mathcal{D}}(\mathcal{A},C)$ using Lemma 3.6.

Data: A set of components and directed wires between them, forming a directed acyclic graph (DAG). Some wires have no beginning component, these are the global inputs. Some wires may have no end component. S is a subset of global inputs. T is a subset of all wires.
Result: Maximum information flow in the workflow
Find (e.g. using breadth-first search) all wires and components through which there is a path from S to T.
Remove all other wires and components.
Set the capacity of each wire to be the maximum entropy of the data that can be sent over the wire (e.g. the number of bits for fixed-length data).
foreach *remaining component* M **do**
  Find its remaining input wires A_M and its remaining output wires C_M.
  Find a bound on $q_M(A_M; C_M)$ as described in Sec. 3.3.4.4.
  Replace the component M with vertices In_M and Out_M so that
    the wires A_M now enter In_M, and
    the wires C_M now begin from Out_M.
  Add an edge w_M from In_M to Out_M with capacity $\varphi(w_M) = q_M(A_M; C_M)$.
end
Add a vertex Source from which the wires S begin.
Add a vertex Sink into which the wires T enter.
Find the maximum flow from Source to Sink.
Return the maximum flow.

Algorithm 2: Maximum Information Flow in a System

Maximum information flow in a workflow. After we have obtained the upper bounds on the mutual information between the inputs and outputs of each component, we use Alg. 2 to find the maximum information flow F in the whole workflow. This is an upper bound on the amount of information that an adversary can leak from S to T. Based on the workflow, and the input and output wires, the algorithm constructs a network (a directed graph, where each arc has been labeled with its capacity, together with distinguished source and sink vertices), such that the maximum flow in this graph is the upper bound that we seek. The following theorem states that F is indeed an upper bound to the amount of information that can be leaked.

Theorem 3.7 (Correctness of Alg. 2)**.** *Suppose that Alg. 2 has been run, finding the maximum flow F in a system. Assume that* $\mathrm{d}(S) \in \mathcal{D}_S$ *and* $\mathrm{d}(\mathcal{G} \setminus S) \in \mathrm{Const}(\mathcal{G} \setminus S)$*. Then* $I(S;T) \leq F$.

Proof. Let C be a minimum cut of the transformed graph in Alg. 2. The inputs and the outputs of a component M in the transformed graph, are A_M and C_M, respectively. In this proof, the occurrences of words like "edge", "path", etc. refer to the transformed graph, not the original graph. W.l.o.g. we can assume that C contains all zero-capacity edges of the transformed graph (because adding edges with zero capacity to the cut does not change the minimality of the cut). Let D be the set of edges outside C from which there is a path to T that does not contain any of the edges in C. Let $e_1, \ldots, e_s$ be the edges in $C \cup D$ in a topological order. Each edge corresponds to either a wire or a component in the original workflow. For each edge e, let

$$o(e) = \begin{cases} C_M & \text{if } e \text{ corresponds to a component } M \\ \{w\} & \text{if } e \text{ corresponds to a wire } w \end{cases}$$

$$c(e) = \begin{cases} \text{the capacity of } M & \text{if } e \in C \text{ and } e \text{ corresponds to a component } M \\ \text{the capacity of } w & \text{if } e \in C \text{ and } e \text{ corresponds to a wire } w \\ 0 & \text{if } e \in D \end{cases}$$

Then we prove by induction that for all $i \le s$,

$$I\left(S; \bigcup_{j=1}^{i} o(e_j)\right) \le \sum_{j=1}^{i} c(e_j)$$

The case $i = 0$ holds because $I(S; \emptyset) = 0$.

Now suppose that

$$I\left(S; \bigcup_{j=1}^{i} o(e_j)\right) \le \sum_{j=1}^{i} c(e_j)$$

holds. Let $Q = \bigcup_{j=1}^{i} o(e_j)$.

First consider the case where $e_{i+1} \in D$ corresponds to a component M. Consider an edge e corresponding to an input wire w of M. If $e \notin C$ then the path obtained by adding e to the beginning of a path from e_{i+1} to T that does not intersect C, is a path from e to T that does not intersect C, thus $e \in D$. Thus $e \in C \cup D$. Because there is path from e to e_{i+1}, e must be earlier in the topological order, i.e. $e = e_k$ for some $k < i + 1$. Because e corresponds to a wire w, $o(e_k) = w$, also $o(e_k) \subseteq Q$, thus $w \in Q$. Thus $A_M \subseteq Q$. Because of topological order, there is no path from M to $Q \setminus A_M$. Thus by Lemma 3.5, $I(S, Q \setminus A_M; C_M | A_M) = 0$. Also $c(e_{i+1}) = 0$. Now

$$I\left(S; \bigcup_{j=1}^{i+1} o(e_j)\right) = I(S; Q \cup C_M) = I(S; Q) + I(S; C_M | Q) \le$$

$$\le I(S; Q) + I(S, Q \setminus A_M; C_M | A_M) = I(S; Q) \le \sum_{j=1}^{i} c(e_j) = \sum_{j=1}^{i+1} c(e_j)$$

Now consider the case where $e_{i+1} \in C$ corresponds to a component M. Because of topological order, there is no path from M to $Q \setminus A_M$. Thus by Lemma 3.5, $I(S, Q \setminus A_M; C_M | A_M) = 0$. Also $c(e_{i+1}) \ge I(A_M; C_M)$. Now

$$I(S; C_M | Q) \le I(S, Q; C_M) \le I(S, Q \cup A_M; C_M) =$$
$$= I(A_M; C_M) + I(S, Q \setminus A_M; C_M | A_M) \le c(e_{i+1})$$
$$I\left(S; \bigcup_{j=1}^{i+1} o(e_j)\right) = I(S; Q \cup C_M) = I(S; Q) + I(S; C_M | Q) \le$$
$$\le \left(\sum_{j=1}^{i} c(e_j)\right) + c(e_{i+1}) = \sum_{j=1}^{i+1} c(e_j)$$

Now consider the case where $e_{i+1} \in D$ corresponds to a wire w. Then there is a path from w to T that does not intersect C. w cannot be a global input because otherwise there would be a path from S to T that does not intersect C, thus it also would not contain zero-capacity edges, thus it would be an augmenting path with positive capacity, contradicting the minimality of the cut C. Thus w is an output of a component M. Consider an edge e corresponding to an input wire w of M. If $e \notin C$ then the path obtained by adding e to the beginning of a path from e_{i+1} to T that does not intersect C, is a path from e to T that does not intersect C, thus $e \in D$. Thus $e \in C \cup D$. Because there is path from e to e_{i+1}, e must be earlier in the topological order, i.e. $e = e_k$ for some $k < i + 1$. Now $w \in o(e_k)$ and $o(e_{i+1}) \subseteq o(e_k) \subseteq Q$. Also $c(e_{i+1}) = 0$. Thus

$$I\left(S; \bigcup_{j=1}^{i+1} o(e_j)\right) = I\left(S; \bigcup_{j=1}^{i} o(e_j)\right) \le \sum_{j=1}^{i} c(e_j) = \sum_{j=1}^{i+1} c(e_j)$$

Now consider the case where $e_{i+1} \in C$ corresponds to a wire w. Then $c(e_{i+1}) \geq H(w)$, the entropy of the value on the wire. Thus

$$I(S;w|Q) \leq I(S,Q;w) = H(w) + H(S,Q) - H(S,Q,w) \leq H(w) \leq c(e_{i+1})$$
$$I\left(S;\bigcup_{j=1}^{i+1} o(e_j)\right) = I(S;Q,w) = I(S;Q) + I(S;w|Q) \leq$$
$$\leq \left(\sum_{j=1}^{i} c(e_j)\right) + c(e_{i+1}) = \sum_{j=1}^{i+1} c(e_j)$$

We have thus proved the induction step for all cases. Now we can estimate $I(S;T)$. Consider any edge e corresponding to a wire in T. If $e \notin C$ then there is a path from e to T that does not intersect C, thus $e \in D$. Thus $e \in C \cup D$. Thus $T \subseteq C \cup D = \bigcup_{j=1}^{s} o(e_j)$.

$$I(S;T) \leq I(S;C \cup D) \leq \sum_{j=1}^{s} c(e_j) = F$$

Here the second inequality holds by the result we proved by induction. The equality holds by the maximum-flow-minimum-cut theorem ($\sum_{j=1}^{s} c(e_j)$ is the value of the minimum cut C). □

3.3.4.6 Completeness of Alg. 2. We can also show the completeness of Alg.2 in some sense, i.e. that under certain conditions, certain (very strong) adversaries can bring the leakage arbitrarily close to the bound F, with arbitrarily small (but positive) error probability.

Suppose that for each port $p \in \mathit{Ports}$, the set Ports also contains ports $p^{(1)}, p^{(2)}, \ldots$ with $\mathrm{V}(p^{(i)}) = \mathrm{V}(p)$. For a set of ports P, let $P^{(1..n)}$ denote the set of ports $\{p^{(i)} \mid p \in P, i \in \{1,\ldots,n\}\}$. For a component M, let $M^{(n)}$ be the component "executing n copies of M in parallel". I.e. the input and output ports of $M^{(n)}$ are $\mathsf{ip}_{M^{(n)}} = \mathsf{ip}_M^{(1..n)}$, and $\mathsf{op}_{M^{(n)}} = \mathsf{op}_M^{(1..n)}$. The function $f_{M^{(n)}}$ takes the n copies of the inputs and independently applies f_M to each copy, resulting in n different sets of outputs.

Let M be a component and P_I, P_O subsets of its input and output ports. Let $f_I : \prod_{p\in P_I} \mathrm{V}(p) \to \mathcal{D}(\prod_{p\in P_I} \mathrm{V}(p))$ and $f_O : \prod_{p\in P_O} \mathrm{V}(p) \to \mathcal{D}(\prod_{p\in P_O} \mathrm{V}(p))$. Let $\mathbf{a}_I \in \prod_{p\in \mathsf{ip}_M \setminus P_I} \mathrm{V}(p)$. Let the mapping $\overline{f_M}$ have the same type as f_M, and be constructed by first applying f_I to the values appearing on P_I, then f_M to the results of f_I and the values $\mathbf{a}_I$ (i.e. the values on ports $\mathsf{ip}_M \setminus P_I$ are ignored), and finally f_O only to the outputs of f_M that would go to ports P_O in M (other outputs pass beside f_O). The *augmentation of M with $P_I, P_O, f_I, f_O, \mathbf{a}_I$* is the component $\mathsf{aug}(P_I, f_I, \mathbf{a}_I; M; f_O, P_O)$ with the same input and output ports as M, and with the function $\overline{f_M}$.

The augmentation of a component is used to "change the encoding" of its inputs and outputs. If the mutual information between the inputs P_I and outputs P_O of M was q, then this is the bound also for the mutual information between the same inputs and outputs of $\mathsf{aug}(P_I, f_I, \mathbf{a}_I; M; f_O, P_O)$.

Let $WF = (\mathcal{M}, \mathcal{W}, \mathsf{s}, \mathsf{t})$ be a workflow. For each component $M \in \mathcal{M}$, let $P_{M;I}$ and $P_{M;O}$ be subsets of ip_M and op_M, respectively. For each n, let $\mathcal{S}^n_{M;I}$ and $\mathcal{S}^n_{M;O}$ be mappings with the following types:

$$\mathcal{S}^n_{M;I} : \quad \prod\nolimits_{p\in P_{M;I}^{(1..n)}} \mathrm{V}(p) \to \mathcal{D}\Big(\prod\nolimits_{p\in P_{M;I}^{(1..n)}} \mathrm{V}(p)\Big)$$
$$\mathcal{S}^n_{M;O} : \quad \prod\nolimits_{p\in P_{M;O}^{(1..n)}} \mathrm{V}(p) \to \mathcal{D}\Big(\prod\nolimits_{p\in P_{M;O}^{(1..n)}} \mathrm{V}(p)\Big) \ .$$

Also, let $\mathcal{S}^n_{M;\mathsf{v}} \in \prod_{p\in \mathsf{ip}_{M^{(n)}} \setminus P_{M;I}^{(1..n)}} \mathrm{V}(p)$. We consider $\mathcal{S}$ to be a function that maps a number n and a component (name) M into a pair of mappings and a tuple of values. We call the tuple of subsets of ports $[(P_{M;I}, P_{M;O})]_{M\in\mathcal{M}}$ the *type* of $\mathcal{S}$. We call $\mathcal{S}$ a *simulator* for WF.

The workflow $WF_{\mathcal{S}}^{(n)}$ intuitively executes n copies of WF, where each component $M^{(n)}$ has been augmented using $\mathcal{S}$. Formally, $WF_{\mathcal{S}}^{(n)} = (\mathcal{M}_n, \mathcal{W}_n, \mathsf{s}, \mathsf{t})$, where

- $\mathcal{M}_n = \{\mathsf{aug}(P_{M;I}, \mathcal{S}^n_{M;I}, \mathcal{S}^n_{M;\mathsf{v}}; M^{(n)}; \mathcal{S}^n_{M;O}, P_{M;O}) \mid M \in \mathcal{M}\}$;
- $\mathcal{W}_n = \{(w, i) \mid w \in \mathcal{W}, i \in \{1, \ldots, n\}\}$;
- $\mathsf{s}(p^{(i)}) = (\mathsf{s}(p), i)$ and $\mathsf{t}(p^{(i)}) = (\mathsf{t}(p), i)$ for all output and input ports of the components in $\mathcal{M}_n$.

Theorem 3.8. *Suppose that Alg. 2 has been run, finding the maximum flow F in the workflow WF. For each component M, let $\mathcal{D}_M$ be the set of allowed probability distributions of A_M, as restricted by the sensitivity declarations. If for each component M, the bound $q_M = q_M^{\mathcal{D}_M}(A_M; C_M)$ found by the algorithm is tight, i.e. there exists $P \in \mathcal{D}_M$ such that if $A_M \sim P$ then $I(A_M; C_M) = q_M$, then for all $\epsilon > 0$, there exists a simulator $\mathcal{S}$ with type $[(A_M, C_M)]_{M \in \mathcal{M}}$, such that for each $\delta > 0$, there exists $n > 0$ such that the workflow $WF^{(n)}_{\mathcal{S}}$ can leak at least $n(F - \epsilon)$ bits of information with the error probability at most δ.*

Proof. Consider a component M. The weight of the edge e corresponding to this component in the flow graph is $q_0 = q_M^{\mathcal{D}}(\mathcal{A}; C)$. Let us run the maximum flow algorithm again with the weight of each edge corresponding to a component reduced by ϵ_0, i.e. $q = q_0 - \epsilon_0$. Then the maximum flow in this modified network is at least $F - K\epsilon_0$ where K is the number of components in the network and F is the flow in the original network. The flow through the edge e determined by the maximum flow algorithm is $f \leq q$.

Let $\mathrm{d}(\mathcal{A}) \in \mathcal{D}$ and $\mathrm{d}(\mathcal{I} \setminus \mathcal{A}) \in \mathrm{Const}(\mathcal{I} \setminus \mathcal{A})$ be such that maximize $I(\mathcal{A}; C)$ in (7). There are n copies of the workflow executed in parallel. The simulator $\mathcal{S}$ consists of pre- and postprocessing tools for each component M. There is a (single) preprocessor $\mathcal{S}^n_{M;A_M}$ before the n copies of M that takes the total of nf bits (assumed to be from the uniform distribution) on the n copies of the wires $\mathcal{A}$ destined to M and encodes them into an n-tuple whose components are each from the distribution $\mathrm{d}(\mathcal{A})$ (not necessarily independent). The tuple of constants $\mathcal{S}^n_{M;\mathsf{v}}$ has been picked from the constant distribution $\mathrm{d}(\mathcal{I} \setminus \mathcal{A})$; these are sent to the n copies of the wires $\mathcal{I} \setminus \mathcal{A}$ destined to M. There is a (single) postprocessor $\mathcal{S}^n_{M;C_M}$ after the n copies of M that takes the n-tuple from the n copies of C and decodes them into a total of nf bits.

By well-known results from information theory, the encoding/decoding (for using a channel with capacity at least $f + \epsilon_0$ for n times) can be chosen in such a way that these Nf bits are with probability at least $1 - \delta_0$ equal to the nf bits that were encoded by the simulator before the n copies of M. The probability that for each component M, the bits sent to the encoder before M are equal to the the bits received from the decoder after M, is at least $1 - K\delta_0$. Thus also the probability that the $n(F - K\epsilon_0)$ bits of the source are equal to the $n(F - K\epsilon_0)$ bits of the sink, is at least $1 - K\delta_0$ (with the variables quantified as follows: $\forall\epsilon_0 \forall\delta_0 \exists n$). We can take $\epsilon = K\epsilon_0$ and $\delta = K\delta_0$ and get that the augmented workflow can leak $n(F - \epsilon)$ bits from the source to the sink with probability at least $1 - \delta$. □

3.3.5 Sensitivity for Components. Often, when studying some process and its semantics, expressed as a function $f : X \to Y$, the elements of X have a lot of structure. Indeed, they may be seen as data structures with several components. When characterizing the sensitivity of f, we would like to do it with respect to different components of its inputs, and say that a certain component affects the outcome of f a lot, while some other component not at all or only a little. When X is a Cartesian product of several sets, then these definitions are not that difficult to come by. But we are interested in a more general case, where the different components are somehow less orthogonal of each other.

In this section, we define component sensitivity and study its composability. Throughout this section, let $(X, d_X(\cdot, \cdot))$ be a metric space defined over a set X, with a metric $d_X(\cdot, \cdot)$.

3.3.5.1 Definitions. We define a component as some equivalence relation on X. It is only important that a component partitions the set X to a set of mutually disjoint sets X/ρ, called *classes*, and we will not use the equivalence relations directly. For components ρ and σ of X, we use $\rho \sqcap \sigma$ to denote the coarsest component that is finer than ρ and σ, i.e. whose equivalence classes are all possible intersections between classes of X/ρ and X/σ. For $x \in X$, we use $[x]_\rho$ to denote the class $X' \in X/\rho$ such that $x \in X'$, which is defined uniquely, since the classes are disjoint. We denote $[m] := \{1, \ldots, m\}$.

Definition 3.10 (Component)**.** A *component* ρ of the set X is an equivalence relation on the set X.

Definition 3.11 (Component composition)**.** Let ρ and σ be two components of X. The component $\rho \sqcap \sigma$ is defined as $X/(\rho \sqcap \sigma) := \{X^\rho \cap X^\sigma \mid X^\rho \in X/\rho, X^\sigma \in X/\sigma\}$.

We extend the distances between elements of X to distances between subsets of X. We use the notation $\tilde{d}_X(\cdot,\cdot)$ for the distance $d_X(\cdot,\cdot)$ extended to subsets of X. Some natural ways to do it are the *max-distance* and the *Hausdorff distance*.

Definition 3.12 (max-distance)**.** Let $X_1, X_2 \subseteq X$. The *max-distance* between X_1 and X_2 is

$$\tilde{d}_X(X_1, X_2) := \max_{x_1 \in X_1, x_2 \in X_2} d_X(x_1, x_2) \ .$$

Definition 3.13 (Hausdorff distance)**.** Let $X_1, X_2 \subseteq X$. The *Hausdorff distance* between X_1 and X_2 is

$$\tilde{d}_X(X_1, X_2) := \max\{\max_{x_1 \in X_1} \min_{x_2 \in X_2} d_X(x_1, x_2), \max_{x_2 \in X_2} \min_{x_1 \in X_1} d_X(x_1, x_2)\} \ .$$

Some particular metrics that we consider are ℓ_p vector norm and ℓ_∞ vector norm.

Definition 3.14 (norm and seminorm)**.** A *seminorm* is a function $\|\cdot\| : V \to \mathbb{R}$ from a vector space V, satisfying the following axioms for all $\vec{x}, \vec{y} \in V$:

- $\|\vec{x}\| \geq 0$;
- $\|\alpha\vec{x}\| = |\alpha| \cdot \|\vec{x}\|$ (implying that $\|\vec{0}\| = 0$);
- $\|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$ (triangle inequality).

Additionally, if $\|\vec{x}\| = 0$ holds only if $\vec{x} = \vec{0}$, then $\|\cdot\|$ is a *norm*.

Some of the most useful norms in practice are ℓ_1 (i.e., sum), ℓ_2 (e.g. geographical distance), and ℓ_∞ (maximum). These are instances of ℓ_p-norms.

Definition 3.15 (ℓ_p-norm)**.** Let $X_i \subseteq \mathbb{R}$, $p \in [1, \infty]$. The ℓ_p-norm of $\vec{x} \in X_1 \times \cdots \times X_n$, denoted $\|\vec{x}\|_p$ is defined as

$$\|\vec{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} .$$

For $p = \infty$, ℓ_∞ is defined as

$$\|\vec{x}\|_\infty = \lim_{p \to \infty} \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} = \max_{i=1}^{n} |x_i| \, .$$

We now introduce some new notions that will be used in this section. We already used $[n] = \{1, \ldots, n\}$. Let the *friend* of $x \in X$ in the set $X' \subseteq X$ be an element $x' \in X'$ that is the closest to x. If there are several closest elements, let all of them be friends. It is possible that there is no closest element.

Definition 3.16 (friend in the set)**.** Let $X' \subseteq X$, $x \in X$. An element $x' \in X'$ is called a *friend* of x in the set X', denoted $x \overset{X'}{\sim} x'$, if

$$x' \in \arg\min_{x' \in X'} d_X(x, x') \ .$$

Two subsets X_1 and X_2 of X are called *friendly* in a subset X' if at least one friend of X_1 in X' belongs to X_2, and each element of X_2 is a friend in X' of some element of X_1.

Definition 3.17 (friendly sets)**.** Subsets $X_1, X_2 \subseteq X$ are called *friendly* in X', denoted $X_1 \overset{X'}{\sim} X_2$, if the following conditions hold:

- $\forall x_1 \in X_1 \ \exists x_2 \in X_2 : x_1 \overset{X'}{\sim} x_2$;
- $\forall x_2 \in X_2 \ \exists x_1 \in X_1 : x_1 \overset{X'}{\sim} x_2$.

We can now give an alternative definition for the Hausdorff distance, expressed through the notion of friend. We will use this definition further in this section.

Definition 3.18 (Hausdorff distance (alternative))**.** Let $X_1, X_2 \subseteq X$. The *Hausdorff distance* between X_1 and X_2 is

$$\tilde{d}_X(X_1, X_2) := \max\{ \max_{x_1 \in X_1, x_2 \in X_2, x_1 \overset{X_2}{\sim} x_2} d_X(x_1, x_2), \max_{x_1 \in X_1, x_2 \in X_2, x_2 \overset{X_1}{\sim} x_1} d_X(x_1, x_2)\} \ .$$

The sensitivity (Def. 3.5) is a global characterization of the function f, describing its worst-case behavior. For more detailed understanding of the behavior, we need some related quantity that is easier to estimate in practice, and from which sensitivity can be derived. We define sensitivity of a function w.r.t. component, which aims to describe sensitivity for inputs that come from two equivalence classes of that component. Defining it for *all* possible pairs would end up in the original definition of sensitivity, so we are trying to find the bounds only for *some* certain pairs of elements. Composition theorems allow us to compute sensitivity of f w.r.t. finer components from sensitivities w.r.t. coarser components. Finally, if we manage to achieve the finest possible component, where each element is in its own equivalence class, it gives us a bound on the general sensitivity of f.

Since we do not know which information exactly will be known for some particular metric space $(X, d_X(\cdot, \cdot))$, what will be the nature of its components, and which distances will be used, we propose several possible definitions. In all definitions below, $c \in \mathbb{R}_+$.

Definition 3.19 (A-sensitivity w.r.t. component)**.** Mapping $f : X \to Y$ is c A-sensitive w.r.t. the component ρ if for all $X_1, X_2 \in X/\rho$ we have

$$\forall x_1 \in X_1, x_2 \in X_2 : \ x_1 \overset{X_2}{\sim} x_2 \implies d_Y(f(x_1), f(x_2)) \leq c \cdot d_X(x_1, x_2) \ .$$

Definition 3.20 (B-sensitivity w.r.t. component)**.** Mapping $f : X \to Y$ is c B-sensitive w.r.t. the component ρ if for all $X_1, X_2 \in X/\rho$ we have

$$\forall x_1 \in X_1, x_2 \in X_2 : \ x_1 \overset{X_2}{\sim} x_2 \implies d_Y(f(x_1), f(x_2)) \leq c \cdot \tilde{d}_X(X_1, X_2) \ .$$

Similar definitions can be used to estimate distances not between single elements, but between subsets of the image of f. For $X' \subseteq X$, we denote $f(X') := \{f(x) \mid x \in X'\}$.

Definition 3.21 (C-sensitivity w.r.t. component)**.** Mapping $f : X \to Y$ is c C-sensitive w.r.t. the component ρ if for all $X_1, X_2 \in X/\rho$ we have

$$\forall X_1' \subseteq X_1, X_2' \subseteq X_2 : \ X_1' \overset{X_2}{\sim} X_2' \implies \tilde{d}_Y(f(X_1'), f(X_2')) \leq c \cdot \tilde{d}_X(X_1', X_2') \ .$$

Definition 3.22 (D-sensitivity w.r.t. component)**.** Mapping $f : X \to Y$ is c D-sensitive w.r.t. the component ρ if for all $X_1, X_2 \in X/\rho$ we have

$$\forall X_1' \subseteq X_1, X_2' \subseteq X_2 : \ X_1' \overset{X_2}{\sim} X_2' \implies \tilde{d}_Y(f(X_1'), f(X_2')) \leq c \cdot \tilde{d}_X(X_1, X_2) \ .$$

Remark 3.1. In some sets, min and max do not exist. If we want to use sup and inf instead of min and max, we could take $x' \in \arg\min_{x' \in \overline{X'}} d_X(x, x')$ instead of $x' \in \arg\min_{x' \in X'} d_X(x, x')$ in Definition 3.16, where $\overline{X'}$ is the closure of X'. However, it may happen that sup and inf of distances are not achievable even in $\overline{X'}$. We would also need to be careful since using closures may eliminate the assumption that equivalence classes do not intersect. A nicer solution would be to define a δ-friend as x' such that $d_X(x, x') - \inf_{x' \in X'} d_X(x, x') \leq \delta$ and ε-sensitivity, which gives an imprecise bound $c \cdot (d_X(x, x') + \varepsilon)$, where ε depends on δ. In this work, we only consider the simpler definition.

3.3.5.2 Properties of Components with Composable Sensitivity. In this section, we give some properties of components that will be needed to make the sensitivity w.r.t. them composable.

If ρ is a component of X, let us call x' a ρ-friend of x if $x \overset{[x']_\rho}{\sim} x'$. The components ρ and σ are called *adjacent* if any two $(\rho \sqcap \sigma)$-friends x and x' have at least one common friend that is ρ-friend of x and σ-friend of x', and that shares its ρ-class with x' and its σ-class with x. This property allows us to reduce $x \overset{[x']_{\rho\sqcap\sigma}}{\sim} x'$ to $x \overset{[x']_\rho}{\sim} x''$ and $x' \overset{[x]_\sigma}{\sim} x''$, which can be analyzed by using known properties (e.g. sensitivity) of the simpler components σ and ρ.

Definition 3.23 (adjacent components)**.** The component pair (ρ, σ) of X is called *adjacent* if for all $x, x' \in X$, $x \overset{[x']_{\rho\sqcap\sigma}}{\sim} x'$, there exists $x'' \in X$ such that:

1. $x \overset{[x']_\rho}{\sim} x''$;
2. $x' \overset{[x]_\sigma}{\sim} x''$.

Definition 3.20 and Definition 3.21 give us upper bounds of the form $\tilde{d}_X([x]_\rho, [x']_\rho)$ for a single component ρ. When we compose the components, we need that the upper bound would depend only on $\tilde{d}_X([x]_{\rho\sqcap\sigma}, [x']_{\rho\sqcap\sigma})$, but this distance can be larger as well as smaller than $\tilde{d}_X([x]_\rho, [x']_\rho)$ and $\tilde{d}_X([x]_\sigma, [x']_\sigma)$. If it is smaller, then the bounds for sensitivities w.r.t. ρ and σ do not provide a reasonable estimation. Hence, we need one more constraint on the components, which ensures that the distance between classes of finer components may only increase.

Definition 3.24 (expanding components)**.** The component pair (ρ, σ) of X is called *expanding* if for all $x, x' \in X$, the following implications hold:

1. $\tilde{d}_X([x]_\rho, [x']_\rho) \leq \tilde{d}_X([x]_{\rho\sqcap\sigma}, [x']_{\rho\sqcap\sigma})$;
2. $\tilde{d}_X([x]_\sigma, [x']_\sigma) \leq \tilde{d}_X([x]_{\rho\sqcap\sigma}, [x']_{\rho\sqcap\sigma})$.

There is another property that can be useful for sensitivity compositions, that can be used as an alternative for the expanding property. One property is neither stronger or weaker than the other one, so we decide to use them both. While the new property looks more specific, its good aspect is that it characterizes a single component, not a component pair. We call a component *equidistant* if, in any two of its classes, all friend pairs have the same distance from each other.

Definition 3.25 (equidistant component)**.** A component ρ of X is called *equidistant* if for all $x, x' \in X$, if $x \overset{[x']_\rho}{\sim} x'$, we have

$$d_X(x, x') = \tilde{d}_X([x]_\rho, [x']_\rho) \ .$$

3.3.5.3 Sensitivity of Compositions. In this section, we first prove some auxiliary lemmas, and then show which conditions need to be satisfied to make it possible to compose sensitivities w.r.t. different components.

Lemma 3.9. *For all $x, x' \in X$, if $x \overset{X_1}{\sim} x'$, then also $x \overset{X_2}{\sim} x'$ for any $X_2 \subseteq X_1$ such that $x' \in X_2$.*

Proof. Let $x, x' \in X$ be such that $x \overset{X_1}{\sim} x'$. Take $x'' \in X_2$ such that $x \overset{X_2}{\sim} x''$. Since there are only less possibilities to choose the minimum from a smaller set, $\min_{z\in X_1} d_X(x, z) \leq \min_{z\in X_2} d_X(x, z)$, and we have $d_X(x, x') \leq d_X(x, x'')$. On the other hand, $x' \in X_2$, so it is a valid friend candidate for x in the set X_2, and it can be only closer to x than x'' is. Hence, $x \overset{X_2}{\sim} x'$. □

Corollary 3.10. *Let ρ be a component of X. For all $x, x' \in X$, if $x \overset{[x']_\rho}{\sim} x'$, then also $x \overset{[x']_{\rho\sqcap\sigma}}{\sim} x'$ for any component σ of X.*

Proof. By definition of component intersection, $[x']_{\rho\sqcap\sigma} \subseteq [x']_\rho$. The classes $[x']_{\rho\sqcap\sigma}$ and $[x']_\rho$ have been chosen in such a way that they both contain x'. Hence, we may apply Lemma 3.9. □

Lemma 3.11. *If $\tilde{d}_X(\cdot,\cdot)$ is either Hausdorff distance or max-distance, then $d_X(x_1,x_2) \le \tilde{d}_X(X_1,X_2)$ for all $X_1, X_2 \subseteq X$, $x_1 \in X_1$, $x_2 \in X_2$, such that $x_1 \overset{X_2}{\sim} x_2$.*

Proof. Let $\tilde{d}_X(\cdot,\cdot)$ be either Hausdorff distance or max-distance.

- For max-distance, $\tilde{d}_X(X_1,X_2) = \max_{x_1\in X_1, x_2\in X_2} d_X(x_1,x_2)$, so $d_X(x_1,x_2) \le \tilde{d}_X(X_1,X_2)$ holds for all $x_1 \in X_1$, $x_2 \in X_2$.
- For Hausdorff distance, $\tilde{d}_X(X_1,X_2) = \max_{i\in\{1,2\},j=3-i}\{\max_{x_i\in X_i, x_j\in X_j, x_i \overset{x_j}{\sim} x_j} d_X(x_i,x_j)\}$, and hence $d_X(x_1,x_2) \le \tilde{d}_X(X_1,X_2)$ holds for all x_1 and x_2 such that $x_1 \overset{X_2}{\sim} x_2$. □

Let us henceforth assume that $\tilde{d}_X(\cdot,\cdot)$ is either Hausdorff distance or max-distance. The distance $\tilde{d}_Y(\cdot,\cdot)$ can in general be arbitrary, and we will mention when some particular instantiation is needed.

Theorem 3.12. *If $\tilde{d}_Y(\cdot,\cdot)$ is Hausdorff distance, then A-sensitivity is equivalent to C- sensitivity, and B-sensitivity is equivalent to D-sensitivity.*

Proof. Let ρ be a component of X, and $X_1, X_2 \in X/\rho$. Let $f : X \to Y$ be c_I I-sensitive, where $\mathrm{I} \in \{\mathrm{A},\mathrm{B},\mathrm{C},\mathrm{D}\}$.

⇐ Let $x_1 \in X_1$, $x_2 \in X_2$. At least for max-distance and Hausdorff distance, we have $d_X(x_1,x_2) = \tilde{d}_X(\{x_1\},\{x_2\})$ and $d_Y(f(x_1),f(x_2)) = \tilde{d}_Y(f(\{x_1\}),f(\{x_2\}))$, Hence, we may always estimate $(\{x_1\},\{x_2\})$ instead of (x_1,x_2). Thus $c_A \le c_C$ and $c_B \le c_D$.

⇒ Let $X_1' \overset{X_2}{\sim} X_2'$. The A- and B-sensitivities give us bounds only for the elements that are friends in X_2, but $\tilde{d}_Y(f(X_1'),f(X_2'))$ may also depend on non-friends of X_1' and X_2', since they may become friends after f is applied to them. We will show that the bounds c_A and c_B of A- and B-sensitivities are still valid. Let $P := \{(x_1,x_2) \mid x_1 \in X_1', x_2 \in X_2', x_1 \overset{X_2}{\sim} x_2\}$. We are able to apply A- and B-sensitivity to all $(x_1,x_2) \in P$.

Since $X_1' \overset{X_2}{\sim} X_2'$, the first coordinates of pairs of P cover the entire set X_1', and the second coordinates of P cover the entire set X_2', possibly with repetitions. By Lemma 3.9, for all $(x_1,x_2) \in P$, we have $x_1 \overset{X_2'}{\sim} x_2$, and hence, by Lemma 3.11, we have $d_X(x_1,x_2) \le \tilde{d}_X(X_1',X_2')$. Applying Lemma 3.11 directly to $x_1 \overset{X_2}{\sim} x_2$, we also have $d_X(x_1,x_2) \le \tilde{d}_X(X_1,X_2)$. There are now two cases possible:

- If at least one $(x_1,x_2) \in P$ is such, that $\tilde{d}_Y(f(X_1'),f(X_2')) \le d_Y(f(x_1),f(x_2))$, then we get a valid bound $\tilde{d}_Y(f(X_1'),f(X_2')) \le d_Y(f(x_1),f(x_2)) \le c_A \cdot d_X(x_1,x_2) \le c_A \cdot \tilde{d}_X(X_1',X_2')$, implying $c_C \le c_A$. Similarly, we have $\tilde{d}_Y(f(X_1'),f(X_2')) \le d_Y(f(x_1),f(x_2)) \le c_B \cdot \tilde{d}_X(X_1,X_2)$ with the implication $c_D \le c_B$.
- Suppose that for all $(x_1,x_2) \in P$ we have $\tilde{d}_Y(f(X_1'),f(X_2')) > d_Y(f(x_1),f(x_2))$. Assuming that all maximums exist, and Hausdorff distance is used, let us take x_1 and x_2 such that $f(x_1) \overset{f(X_2')}{\sim} f(x_2)$ and $\tilde{d}_Y(f(X_1'),f(X_2')) = d_Y(f(x_1),f(x_2))$. Since the first coordinates of pairs of P cover the entire set X_1', there is $x_2' \in X_2'$ such that $(x_1,x_2') \in P$. By assumption, $\tilde{d}_Y(f(X_1'),f(X_2')) > d_Y(f(x_1),f(x_2'))$, which implies $d_Y(f(x_1),f(x_2)) > d_Y(f(x_1),f(x_2'))$. However, since $f(x_1) \overset{f(X_2')}{\sim} f(x_2)$, the element $f(x_2') \in f(X_2')$ cannot be strictly closer to $f(x_1)$ than $f(x_2)$ is, which is a contradiction. □

We note that the ⇒ implication does not hold in general if $\tilde{d}_Y(\cdot,\cdot)$ max-distance, since x_1 and x_2 that achieve $\tilde{d}_Y(f(X_1'),f(X_2')) = d_Y(f(x_1),f(x_2))$ are not necessarily friends, and the equality $\tilde{d}_Y(f(x_1),f(x_2)) > d_Y(f(x_1),f(x_2'))$ does not give any contradictions.

Example 3.3. A tiny counterexample would be $X = Y = \mathbb{R}$, $d_X(x,y) = d_Y(x,y) = |x-y|$, $X_1' = \{0,11\}$, $X_2' = \{2,9\}$, and $f(x) := x \bmod 2$. Since 0 is closer to 2, and 11 is closer to 9, using A- and B-sensitivity, we could give bounds for $d_Y(f(0),f(2)) = d_Y(0,0)$ and $d_Y(f(11),f(9)) = d_Y(1,1) = 0$, for

which an arbitrarily small c is suitable. However, for max-distance we have $\tilde{d}_Y(f(\{0,11\}), f(\{2,9\})) = \tilde{d}_Y(\{0,1\},\{0,1\}) = 1$, and $\tilde{d}_X(\{0,11\},\{2,9\}) \leq 9$ (whenever Hausdorff distance or max-distance is used for $d_X(\cdot,\cdot)$), which allows only $c \geq \frac{1}{9}$.

Theorem 3.13. *Let (ρ,σ) be a adjacent component pair. Let the mapping $f : X \to Y$ be c_ρ and c_σ A-sensitive w.r.t. ρ and σ respectively. Then, f is $(c_\rho + c_\sigma)$ A-sensitive w.r.t $\rho \sqcap \sigma$.*

Proof. Let $x, x' \in X$ be such that $x \overset{[x']_{\rho\sqcap\sigma}}{\sim} x'$. We need to show that $d_Y(f(x), f(x')) \leq (c_\rho + c_\sigma) \cdot d_X(x,x')$. Since (ρ,σ) is a adjacent pair, there exists $x'' \in X$ such that $x \overset{[x']_\rho}{\sim} x''$ and $x' \overset{[x]_\sigma}{\sim} x''$. Since $x'' \in [x']_\rho$ and $x'' \in [x]_\sigma$, we have $[x'']_\rho = [x']_\rho$, $[x'']_\sigma = [x]_\sigma$, and $x \overset{[x'']_\rho}{\sim} x''$, $x' \overset{[x'']_\sigma}{\sim} x''$, so we can apply A-sensitivity:

- $d_Y(f(x), f(x'')) \leq c_\rho \cdot d_X(x,x'')$;
- $d_Y(f(x''), f(x')) \leq c_\sigma \cdot d_X(x'',x')$.

Since $x \overset{[x']_\rho}{\sim} x''$, the element x' cannot be closer to x than x'' is, so $d_X(x,x'') \leq d_X(x,x')$.
Since $x' \overset{[x]_\sigma}{\sim} x''$, the element x cannot be closer to x' than x'' is, so $d_X(x'',x') \leq d_X(x,x')$.
We now estimate $d_Y(f(x), f(x'))$ from above, using the triangle inequality.

$$\begin{aligned} d_Y(f(x), f(x')) &\leq d_Y(f(x), f(x'')) + d_Y(f(x''), f(x')) \\ &\leq c_\rho \cdot d_X(x,x'') + c_\sigma \cdot d_X(x'',x') \\ &\leq c_\rho \cdot d_X(x,x') + c_\sigma \cdot d_X(x,x') \\ &= (c_\rho + c_\sigma) \cdot d_X(x,x') \ . \end{aligned}$$

□

Corollary 3.14. *Let (ρ,σ) be an adjacent pair of equidistant components. Let the mapping $f : X \to Y$ be c_ρ and c_σ B-sensitive w.r.t. ρ and σ respectively. Then, f is $(c_\rho + c_\sigma)$ B-sensitive w.r.t $\rho \sqcap \sigma$.*

Proof. Let $x, x' \in X$, $x \overset{[x']_{\rho\sqcap\sigma}}{\sim} x'$. Since ρ and σ are equidistant, B-sensitivity (Definition 3.20) implies A-sensitivity (Definition 3.19), so f is c_ρ and c_σ A-sensitive w.r.t. ρ and σ respectively. Applying Theorem 3.13, we get $d_Y(f(x), f(x')) \leq (c_\rho + c_\sigma) \cdot d_X(x,x')$. By Lemma 3.11, $d_X(x,x') \leq \tilde{d}_X([x]_\rho, [x']_\rho)$, and this gives us $d_Y(f(x), f(x')) \leq (c_\rho + c_\sigma) \cdot \tilde{d}_X([x]_\rho, [x']_\rho)$, which is the definition of B-sensitivity. □

Theorem 3.15. *Let (ρ,σ) be an adjacent expanding component pair. Let the mapping $f : X \to Y$ be c_ρ and c_σ B-sensitive w.r.t. ρ and σ respectively. Then, f is $(c_\rho + c_\sigma)$ B-sensitive w.r.t $\rho \sqcap \sigma$.*

Proof. Constructing x'' from x and x' as in the proof of Theorem 3.13, we we can apply B-sensitivity of f to the pairs (x,x'') and (x'',x') separately, getting the following inequalities:

- $d_Y(f(x), f(x'')) \leq c_\rho \cdot \tilde{d}_X([x]_\rho, [x'']_\rho) = c_\rho \cdot \tilde{d}_X([x]_\rho, [x']_\rho)$;
- $d_Y(f(x''), f(x')) \leq c_\sigma \cdot \tilde{d}_X([x'']_\sigma, [x']_\sigma) = c_\sigma \cdot \tilde{d}_X([x]_\sigma, [x']_\sigma)$.

Since ρ and σ are expanding, we have $\tilde{d}_X([x]_\rho, [x']_\rho) \leq \tilde{d}_X([x]_{\rho\sqcap\sigma}, [x']_{\rho\sqcap\sigma})$ and $\tilde{d}_X([x]_\sigma, [x']_\sigma) \leq \tilde{d}_X([x]_{\rho\sqcap\sigma}, [x']_{\rho\sqcap\sigma})$.

We now estimate $d_Y(f(x), f(x'))$ from above, using the triangle inequality.

$$\begin{aligned} d_Y(f(x), f(x')) &\leq d_Y(f(x), f(x'')) + d_Y(f(x''), f(x')) \\ &\leq c_\rho \cdot \tilde{d}_X([x]_\rho, [x']_\rho) + c_\sigma \cdot \tilde{d}_X([x]_\sigma, [x']_\sigma) \\ &\leq c_\rho \cdot \tilde{d}_X([x]_{\rho\sqcap\sigma}, [x']_{\rho\sqcap\sigma}) + c_\sigma \cdot \tilde{d}_X([x]_{\rho\sqcap\sigma}, [x']_{\rho\sqcap\sigma}) \\ &= (c_\rho + c_\sigma) \cdot \tilde{d}_X([x]_{\rho\sqcap\sigma}, [x']_{\rho\sqcap\sigma}) \ . \end{aligned}$$

□

Theorem 3.16. *Let (ρ,σ) be an adjacent component pair. Let the mapping $f : X \to Y$ be c_ρ and c_σ C-sensitive w.r.t. ρ and σ respectively. Then, f is $(c_\rho + c_\sigma)$ C-sensitive w.r.t $\rho \sqcap \sigma$.*

Proof. Let $X_1, X_2 \in X/(\rho \sqcap \sigma)$. Let $X_1' \subseteq X_1$ and $X_2' \subseteq X_2$ be such that $X_1' \overset{X_2}{\sim} X_2'$. We need to show that $\tilde{d}_Y(f(X_1'), f(X_2')) \leq (c_\rho + c_\sigma) \cdot \tilde{d}_X(X_1', X_2')$.

Since every class of $X/(\rho \sqcap \sigma)$ is an intersection of two classes of X/ρ and X/σ, for $i \in \{1, 2\}$, we can write $X_i = X_i^\rho \cap X_i^\sigma$, where $X_i^\rho \in X/\rho$ and $X_i^\sigma \in X/\sigma$. Hence, we can represent $X_i' \subseteq X_i$ as $X_i' = X_i'^\rho \cap X_i'^\sigma$, where $X_i'^\rho \subseteq X_i^\rho$ and $X_i'^\sigma \subseteq X_i^\sigma$.

Let $x \in X_1'$. Since $X_1' \overset{X_2}{\sim} X_2'$, there exists $x' \in X_2'$ such that $x \overset{X_2}{\sim} x'$. Since $X_2 \in X/(\rho \sqcap \sigma)$, and (ρ, σ) is an adjacent pair, there exists x'' such that $x \overset{X_2^\rho}{\sim} x''$ and $x' \overset{X_1^\sigma}{\sim} x''$. Let $X'' := \{x'' \mid x \in X_1'\}$ be the set of all such x'' that exist for all elements of X_1' (if there are several such elements for the same x, take all of them, so that a suitable $x'' \in X''$ would exist for all $x' \in X_2'$).

Since X'' is constructed in such a way that $\forall x \in X_1' \; \exists x'' \in X'' : \; x \overset{X_2^\rho}{\sim} x''$, and there are no other elements in X'', we have $X_1' \overset{X_2^\rho}{\sim} X''$, so we can apply C-sensitivity of f to them. Similarly, since X_2' by assumption contains only friends of X_1' and no other elements, we have $\forall x' \in X_2' \; \exists x'' \in X'' : \; x' \overset{X_1^\sigma}{\sim} x''$, and since there are no elements in X'' for which such x' does not exist, we have $X_2' \overset{X_1^\sigma}{\sim} X''$, so we can apply C-sensitivity of f:

- $\tilde{d}_Y(f(X_1'), f(X'')) \leq c_\rho \cdot \tilde{d}_X(X_1', X'')$;
- $\tilde{d}_Y(f(X''), f(X_2')) \leq c_\sigma \cdot \tilde{d}_X(X'', X_2')$.

Consider any pair $x \in X_1'$, $x'' \in X''$, such that $x \overset{X_2^\rho}{\sim} x''$. Since $x \overset{X_2^\rho}{\sim} x''$, no element $x' \in X_2' \subseteq X_2^\rho$ can be closer to x than x'' is, so $d_X(x, x'') \leq d_X(x, x')$ for all $x' \in X_2'$. This inequality holds for all pairs of friends $x \in X_1'$, $x'' \in X''$, and hence also for the pair that gives the maximal distance, so $\tilde{d}_X(X_1', X'') \leq d_X(x, x')$. On the other hand, it holds for all $x' \in X_2'$, and hence also for x' such that $x \overset{X_2'}{\sim} x'$, and by Lemma 3.11 we have $d_X(x, x') \leq \tilde{d}_X(X_1', X_2')$. Putting all together, we get $\tilde{d}_X(X_1', X'') \leq \tilde{d}_X(X_1', X_2')$. Similarly, we get $\tilde{d}_X(X'', X_2') \leq \tilde{d}_X(X_1', X_2')$.

We now estimate $\tilde{d}_Y(f(X_1'), f(X_2'))$ from above, using the triangle inequality.

$$\begin{aligned}
\tilde{d}_Y(f(X_1'), f(X_2')) &\leq \tilde{d}_Y(f(X_1'), f(X'')) + \tilde{d}_Y(f(X''), f(X_2')) \\
&\leq c_\rho \cdot \tilde{d}_X(X_1', X'') + c_\sigma \cdot \tilde{d}_X(X'', X_2') \\
&\leq c_\rho \cdot \tilde{d}_X(X_1', X_2') + c_\sigma \cdot \tilde{d}_X(X_1', X_2') \\
&= (c_\rho + c_\sigma) \cdot \tilde{d}_X(X_1', X_2') \; .
\end{aligned}$$

□

Corollary 3.17. *Let (ρ, σ) be an adjacent pair of equidistant components. Let the mapping $f : X \to Y$ be c_ρ and c_σ D-sensitive w.r.t. ρ and σ respectively. Then, f is $(c_\rho + c_\sigma)$ D-sensitive w.r.t $\rho \sqcap \sigma$.*

Proof. Similarly to Corollary 3.14, since the components are equidistant, we have $d_X(x, x') = \tilde{d}_X(X_1, X_2)$ for all $x \in X_1$, $x' \in X_2$ such that $x \overset{X_2}{\sim} x'$, and $X_1, X_2 \in X/\rho$ or $X_1, X_2 \in X/\sigma$. Hence, also $\tilde{d}_X(X_1', X_2') = \tilde{d}_X(X_1, X_2)$ for $X_1' \subseteq X_1$ and $X_2' \subseteq X_2$ such that $X_1' \overset{X_2}{\sim} X_2'$, so C-sensitivity of ρ and σ implies their D-sensitivity. Applying Theorem 3.16, we get $\tilde{d}_Y(f(X_1'), f(X_2')) \leq (c_\rho + c_\sigma) \cdot \tilde{d}_X(X_1', X_2')$. Since we have $X_1' \overset{X_2}{\sim} X_2'$, every element $x \in X_1'$ has a friend in X_2' that is also a friend in X_2 and hence, by Lemma 3.11, is at most $\tilde{d}_X(X_1, X_2)$ far away from it. This gives us $\tilde{d}_X(X_1', X_2') \leq \tilde{d}_X(X_1, X_2)$. □

Theorem 3.18. *Let (ρ, σ) be an adjacent expanding component pair. Let the mapping $f : X \to Y$ be c_ρ and c_σ D-sensitive w.r.t. ρ and σ respectively. Then, f is $(c_\rho + c_\sigma)$ D-sensitive w.r.t $\rho \sqcap \sigma$.*

Proof. Constructing X'' from X_1' and X_2' as in the proof of Theorem 3.16, we we can apply C-sensitivity of f to the pairs (X_1', X'') and (X_2', X'') separately, getting the following inequalities:

- $\tilde{d}_Y(f(X_1'), f(X'')) \leq c_\rho \cdot \tilde{d}_X(X_1^\rho, X_2^\rho)$;

- $\tilde{d}_Y(f(X''), f(X'_2)) \leq c_\sigma \cdot \tilde{d}_X(X_1^\sigma, X_2^\sigma)$.

Since ρ and σ are expanding, we have $\tilde{d}_X(X_1^\rho, X_2^\rho) \leq \tilde{d}_X(X_1^\rho \cap X_1^\sigma, X_2^\rho \cap X_2^\sigma)$, and $\tilde{d}_X(X_1^\sigma, X_2^\sigma) \leq \tilde{d}_X(X_1^\rho \cap X_1^\sigma, X_2^\rho \cap X_2^\sigma)$. By construction, $\tilde{d}_X(X_1^\rho \cap X_1^\sigma, X_2^\rho \cap X_2^\sigma) = \tilde{d}_X(X_1, X_2)$.

We now estimate $\tilde{d}_Y(f(X'_1), f(X'_2))$ from above, using the triangle inequality.

$$\begin{aligned}
\tilde{d}_Y(f(X'_1), f(X'_2)) &\leq \tilde{d}_Y(f(X'_1), f(X'')) + \tilde{d}_Y(f(X''), f(X'_2)) \\
&\leq c_\rho \cdot \tilde{d}_X(X_1^\rho, X_2^\rho) + c_\sigma \cdot \tilde{d}_X(X_1^\sigma, X_2^\sigma) \\
&\leq c_\rho \cdot \tilde{d}_X(X_1, X_2) + c_\sigma \cdot \tilde{d}_X(X_1, X_2) \\
&= (c_\rho + c_\sigma) \cdot \tilde{d}_X(X_1, X_2) \ .
\end{aligned}$$

□

3.3.5.4 Example of Vectors with Independent Coordinates. We now consider metric spaces $(X, d_X(\cdot,\cdot))$ and components ρ of certain form.

- $X := X_1 \times \cdots \times X_n$;
- $d_X(\vec{x}, \vec{x'}) := \|\vec{x} - \vec{x'}\|_p$ or $d_X(\vec{x}, \vec{x'}) := \|\vec{x} - \vec{x'}\|_\infty$;
- $\tilde{d}_X(X_1, X_2)$ is the Hausdorff distance;
- ρ_i is such that $X_a \in X/\rho_i$ is defined as $X_a := \{(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n) \mid x_j \in X_j\}$.

In this section, we will consider components of the form $\rho := \bigsqcap_{i\in\mathcal{I}} \rho_i$. Since the norms ℓ_p and ℓ_∞ do not depend on the ordering of the coordinates X_i, without loss of generality, let $\mathcal{I} = [m]$ for some $m \leq n$. Let the classes of $X/\bigsqcap_{i\in[m]} \rho_i$ be denoted by $X_{(a_i)_{i\in[m]}} = \{(a_1, \ldots, a_m, x_{m+1}, \ldots, x_n) \mid x_j \in X_j, m < j \leq n\}$.

Lemma 3.19. *Let $\rho = \bigsqcap_{i\in[m]} \rho_i$; $X_{(a_i)_{i\in[m]}}, X_{(a'_i)_{i\in[m]}} \in X/\rho$. For $\vec{x} = (a_1, \ldots, a_m, x_{m+1}, \ldots, x_n) \in X_{(a_i)_{i\in[m]}}$, if $\vec{x'} = (a'_1, \ldots, a'_m, x_{m+1}, \ldots, x_n)$, then $\vec{x} \overset{X_{(a'_i)_{i\in[m]}}}{\sim} \vec{x'}$.*

Proof. First, let us assume that ℓ_p norm is used. We get $d_X(\vec{x}, \vec{x'}) = (\sum_{i=1}^m |a'_i - a_i|^p + \sum_{i=m+1}^n |x_i - x_i|^p)^{1/p} = (\sum_{i=1}^m |a'_i - a_i|^p)^{1/p}$. We cannot achieve a better bound since a'_i are fixed, and taking $x'_i \neq x_i$ in $\vec{x'}$ may only increase the absolute value. Hence, $\vec{x'}$ minimizes $d_X(\vec{x}, \vec{x'})$ in $X_{(a'_i)_{i\in[m]}}$, so $\vec{x} \overset{X_{(a'_i)_{i\in[m]}}}{\sim} \vec{x'}$.

Similarly, if ℓ_∞ norm is used, then $d_X(\vec{x}, \vec{x'}) = \max(\max_{i=1}^m |a'_i - a'_i|, \max_{i=m+1}^n |x_i - x_i|) = \max_{i=1}^m |a'_i - a_i|$, and taking $x'_i \neq x_i$ in $\vec{x'}$ may only increase the maximum. □

Proposition 3.20. *Any component of the form $\rho = \bigsqcap_{i\in[m]} \rho_i$ is equidistant. In particular, $\tilde{d}_X(X_{(a_i)_{i\in[m]}}, X_{(a'_i)_{i\in[m]}}) = (\sum_{i=1}^m |a'_i - a_i|^p)^{1/p}$ for ℓ_p norm, and $\tilde{d}_X(X_{(a_i)_{i\in[m]}}, X_{(a'_i)_{i\in[m]}}) = \max_{i=1}^m |a'_i - a_i|$ for ℓ_∞ norm.*

Proof. Let $X_{(a_i)_{i\in[m]}}, X_{(a'_i)_{i\in[m]}} \in X/\rho$. For any element $\vec{x} = (a_1, \ldots, a_m, x_{m+1}, \ldots, x_n) \in X_{(a_i)_{i\in[m]}}$, we can take $\vec{x'} = (a'_1, \ldots, a'_m, x_{m+1}, \ldots, x_n) \in X_{(a'_i)_{i\in[m]}}$. By Lemma 3.19, $\vec{x} \overset{X_{(a'_i)_{i\in[m]}}}{\sim} \vec{x'}$.

We have $d_X(\vec{x}, \vec{x'}) = (\sum_{i=1}^m |a'_i - a_i|^p + \sum_{i=m+1}^n |x_i - x_i|^p)^{1/p} = (\sum_{i=1}^m |a'_i - a_i|^p)^{1/p}$. Since such $\vec{x'}$ exists for all $\vec{x} \in X_{(a_i)_{i\in[m]}}$, we also have $\max_{\vec{x}\in X_{(a_i)_{i\in[m]}}} \min_{\vec{x'}\in X_{(a'_i)_{i\in[m]}}} d_X(\vec{x}, \vec{x'}) = (\sum_{i=1}^m |a'_i - a_i|^p)^{1/p}$. The same holds if we swap $X_{(a_i)_{i\in[m]}}$ and $X_{(a'_i)_{i\in[m]}}$. By definition of Hausdorff distance, $\tilde{d}_X(X_{(a_i)_{i\in[m]}}, X_{(a'_i)_{i\in[m]}}) = (\sum_{i=1}^m |a'_i - a_i|^p)^{1/p}$.

Similarly, if ℓ_∞ norm is used, then $\max(\max_{i=1}^m |a'_i - a'_i|, \max_{i=m+1}^n |x_i - x_i|) = \max_{i=1}^m |a'_i - a_i|$ for all $\vec{x} \in X_{(a_i)_{i\in[m]}}$, so $\tilde{d}_X(X_{(a_i)_{i\in[m]}}, X_{(a'_i)_{i\in[m]}}) = \max_{i=1}^m |a'_i - a'_i|$. □

Proposition 3.21. *The components $\rho := \bigsqcap_{i\in[m]} \rho_i$ and $\sigma := \rho_{m+1}$ are adjacent.*

Proof. Let $X_{(a_i)_{i\in[m]}}, X_{(a'_i)_{i\in[m]}} \in X/\rho$, and let $X_a, X_{a'} \in X/\sigma$. For any $\vec{x} = (a_1, \ldots, a_m, a, x_{m+2}, \ldots, x_n) \in X_{(a_i)_{i\in[m]}} \cap X_a$ and $\vec{x'} = (a'_1, \ldots, a'_m, a', x_{m+2}, \ldots, x_n) \in X_{(a'_i)_{i\in[m]}} \cap X_{a'}$, we can define $\vec{x''} \in X_{(a'_i)_{i\in[m]}} \cap X_a$ as $\vec{x''} = (a'_1, \ldots, a'_m, a, x_{m+2}, \ldots, x_n)$. By Lemma 3.19, we have $\vec{x} \overset{X_{(a'_i)_{i\in[m]}}}{\sim} \vec{x''}$ and $\vec{x} \overset{X_a}{\sim} \vec{x''}$. □

Corollary 3.22. *For all $i \in [n]$, let c_i be I-sensitivity of a function f w.r.t. the component ρ_i, for* $\mathrm{I} \in \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$. *Then $\sum_{i\in\mathcal{I}} c_i$ is the I-sensitivity of f w.r.t. the component $\bigsqcap_{i\in\mathcal{I}} \rho_i$ for all $\mathcal{I} \subseteq [n]$.*

Proof. Prop. 3.20 and Prop. 3.21 together give us the necessary preconditions for applying Theorem 3.13, Corollary 3.14, Theorem 3.16, Corollary 3.17 for all variants of sensitivity definition. □

3.3.5.5 Example of Vectors with Linearly Dependent Coordinates. We consider metric spaces $(\hat{X}, d_X(\cdot,\cdot))$ of the same form as in Sec. 3.3.5.4, with the following additional conditions.

- $\forall i, j:\ X_i = X_j =: Z$;
- There is a matrix $\mathbf{A} \in Z^{p\times n}$ for $p \le n$, and a vector $\vec{b} \in Z^p$ such that, $\forall \vec{x} \in \hat{X}:\ \mathbf{A}\vec{x} = \vec{b}$.

Similarly to Sec. 3.3.5.4, we consider components of the form $\hat{\rho} := \bigsqcap_{i\in\mathcal{I}} \hat{\rho}_i$, where $\hat{\rho}_i$ is the analog of ρ_i, defined over $\hat{X}$. For equivalence classes, we will use the notation

$$\hat{X}_{(a_i)_{i\in[m]}} = \{(a_1, \ldots, a_m, x_{m+1}, \ldots, x_n) \mid x_j \in X_j, m < j \le n, \mathbf{A}\vec{x} = \vec{b}\}$$

to discern them from the analogous classes of independent components.

Let $\vec{x} = \vec{x_1}|\vec{x_2}$ denote that $\vec{x}$ is a concatenation of vectors $\vec{x_1}$ and $\vec{x_2}$. For matrices, $\mathbf{A} = \mathbf{A_1}|\mathbf{A_2}$ denotes concatenation of rows.

Lemma 3.23. *Without loss of generality, let $\mathcal{I} = [m]$. Let $\hat{\rho} = \bigsqcap_{i\in[m]} \hat{\rho}_i$, and $\hat{X}_{(a_i)_{i\in[m]}}, \hat{X}_{(a'_i)_{i\in[m]}} \in \hat{X}/\hat{\rho}$. Let $\vec{x} \in \hat{X}$ be such that $x_i = a_i$ for $i \in [m]$, and $\mathbf{A}\vec{x} = \vec{b}$. Let $\vec{x'} \in \hat{X}$ be such that $\vec{x} \overset{X_{(a'_i)_{i\in[m]}}}{\sim} \vec{x'}$.*

Let $\mathbf{A} = \mathbf{A_1}|\mathbf{A_2}$, where $\mathbf{A_1}$ are the first m columns and $\mathbf{A_2}$ are the rest. We can write $\vec{x'} = \vec{x} + \vec{z}$, where $\vec{z} = \vec{z_1}|\vec{z_2}$, $\vec{z_1} = (\vec{a'} - \vec{a})$, and $\vec{z_2}$ is an optimal solution to the task

$$\textit{minimize}\|\vec{z_2}\|,\ \textit{subject to}\mathbf{A_2}\vec{z_2} = \mathbf{A_1}(\vec{a} - \vec{a'})\ .$$

Proof. In order to find the vector $\vec{x'}$ that is the closest to $\vec{x}$, we need to solve the following task:

$$\text{minimize}\|\vec{x'} - \vec{x}\|,\ \text{subject to}\mathbf{A}\vec{x'} = \vec{b}\ .$$

We would like to get rid of x_j that make this quantity dependent on the particular instance $\vec{x}$. We may introduce variables $\vec{z} := \vec{x'} - \vec{x}$. This gives us linear constraints $\mathbf{A}(\vec{x} + \vec{z}) = \vec{b}$. Since by assumption $\mathbf{A}\vec{x} = \vec{b}$, this is equivalent to $\mathbf{A}\vec{z} = \vec{0}$. We get the task

$$\text{minimize}\|\vec{z}\|,\ \text{subject to}\mathbf{A}\vec{z} = \vec{0}\ .$$

We can rewrite $\mathbf{A}\vec{z}$ as $(\mathbf{A_1}|\mathbf{A_2})(\vec{z_1}|\vec{z_2}) = \mathbf{A_1}\vec{z_1} + \mathbf{A_2}\vec{z_2}$. For $j \in [m]$, we have $z_j = (a'_j - a_j)$, so $\mathbf{A_1}\vec{z_1} = \mathbf{A_1}(\vec{a'} - \vec{a})$. Substituting this value into the constraints, we get the statement of this lemma. □

Like in the case of independent components, we could first define sensitivity for each single component, and then define their composition. However, it is no longer reasonable to define it for friends as in Sec. 3.3.5.4: due to the constraints, the closest vectors may have very different entries, and we might be unable to define sensitivity even for single components that we could use as building blocks. Moreover, using the same definitions as in Sec. 3.3.5.4, we will not achieve the necessary preconditions for the composition theorems, since dependent components are in general not adjacent.

Instead of defining new theory for dependent components, let us compute upper bounds directly from the theory of independent components. We assume that we only know how to compute initial sensitivity for *independent* components ρ_j, i.e. we can give upper bounds only for $d_X(f(x), f(x'))$ where $x_i = x'_i$ for all $i \ne j$. We use them to construct sensitivities for *dependent* components $\hat{\rho}_j$. We will derive the bounds not for friends, but for *match*es – the vectors that have maximal amount of similar coordinates. Let $d_{Ham}(\vec{x}, \vec{x'})$ denote the *Hamming distance* of $\vec{x}$ and $\vec{x'}$, i.e. the number of their different coordinates.

Definition 3.26 (match in the set)**.** Let $\hat{X}' \subseteq \hat{X}$, $\vec{x} \in \hat{X}$. An element $\vec{x'} \in \hat{X}'$ is called a *match* of $\vec{x}$ in the set $\hat{X}'$, denoted $\vec{x} \overset{\hat{X}'}{\approx} \vec{x'}$, if

$$\vec{x'} \in \arg\min_{\vec{x'} \in \hat{X}'} d_{Ham}(\vec{x}, \vec{x'}) \ .$$

Definition 3.27 (matching sets)**.** Subsets $X_1, X_2 \subseteq X$ are called *matching* in X', denoted $X_1 \overset{X'}{\approx} X_2$, if the following conditions hold:

- $\forall \vec{x_1} \in X_1 \ \exists \vec{x_2} \in X_2 : \vec{x_1} \overset{X'}{\approx} \vec{x_2}$;
- $\forall \vec{x_2} \in X_2 \ \exists \vec{x_1} \in X_1 : \vec{x_1} \overset{X'}{\approx} \vec{x_2}$;

The notions of a friend and a match (as well as friendly sets and matching sets) are equivalent for vectors with independent components, but they are different in the case of dependent components.

Lemma 3.24. *If* $\vec{x} \overset{\hat{X}_{(a_i)_{i\in[m]}}}{\approx} \vec{x'}$*, then there are at least* $\max(0, n-p-m)$ *equal entries in* $\vec{x}$ *and* $\vec{x'}$*.*

Proof. Since $\vec{x'} \in \hat{X}_{(a_i)_{i\in[m]}}$, we already have $x'_i = a_i$ for $i \in [m]$, and it may happen that $a_i \neq x_i$ for all $i \in [m]$, so in general we do not get equal entries from here. The linear equation system has degree of freedom $n-p$, so even after m entries are fixed, if $p+m < n$, then there are at least $n-p-m$ entries whose value can be set to arbitrary, and $x'_j = x_j$ can be taken without violating the constraints. The remaining $\min(n-m, p)$ entries are uniquely determined by the constraints and the other entries. □

Proposition 3.25. *Any component of the form* $\hat{\rho} = \bigsqcap_{i\in[m]} \hat{\rho}_i$ *is equidistant.*

Proof. Let $\hat{X}_{(a_i)_{i\in[m]}}, \hat{X}_{(a'_i)_{i\in[m]}} \in \hat{X}/\hat{\rho}$. For any element $\vec{x} = (a_1, \ldots, a_m, x_{m+1}, \ldots, x_n) \in \hat{X}_{(a_i)_{i\in[m]}}$, we can take $\vec{x'} \in \hat{X}$ such that $\vec{x} \overset{\hat{X}_{(a'_i)_{i\in[m]}}}{\sim} \vec{x'}$, using the construction of Lemma 3.23. As in the proof of Prop. 3.20, it is only important that $d_X(\vec{x}, \vec{x'})$ depends only on the classes $\hat{X}_{(a_i)_{i\in[m]}}$, $\hat{X}_{(a'_i)_{i\in[m]}}$, and on the matrix $\mathbf{A}$ that is fixed, so it can be expressed as some quantity $g(\mathbf{A}, (a_i, a'_i)_{i\in[m]})$ that does not depend on any other entries of $\vec{x}$ and $\vec{x'}$. Since such $\vec{x'}$ exists for all $\vec{x} \in \hat{X}_{(a_i)_{i\in[m]}}$, we also have $\max_{\vec{x}\in\hat{X}_{(a_i)_{i\in[m]}}} \min_{\vec{x'}\in\hat{X}_{(a'_i)_{i\in[m]}}} d_X(\vec{x}, \vec{x'}) = g(\mathbf{A}, (a_i, a'_i)_{i\in[m]})$, and also $\max_{\vec{x'}\in\hat{X}_{(a'_i)_{i\in[m]}}} \min_{\vec{x}\in X_{(a_i)_{i\in[m]}}} = g(\mathbf{A}, (a_i, a'_i)_{i\in[m]})$. By definition of the Hausdorff distance, we get $\tilde{d}_X(\hat{X}_{(a_i)_{i\in[m]}}, \hat{X}_{(a'_i)_{i\in[m]}}) = g(\mathbf{A}, (a_i, a'_i)_{i\in[m]})$. □

Proposition 3.26 (A-sensitivity w.r.t. component)**.** *Let mapping* $f : X \to Y$ *be* c_i *A-sensitive w.r.t. the component* ρ_i *for all* $i \in \{1, \ldots, n\}$*. Let* $\hat{\rho} := \bigsqcap_{i\in\mathcal{I}} \hat{\rho}_i$*. For all* $\hat{X}_1, \hat{X}_2 \in \hat{X}/\hat{\rho}$ *we have*

$$\forall \vec{x_1} \in \hat{X}_1, \vec{x_2} \in \hat{X}_2 : \ \vec{x_1} \overset{\hat{X}_2}{\approx} \vec{x_2} \implies d_Y(f(\vec{x_1}), f(\vec{x_2})) \leq \left(p \cdot \max_{i\in[n]\setminus\mathcal{I}} (c_i) + \sum_{i\in\mathcal{I}} c_i \right) \cdot d_X(\vec{x_1}, \vec{x_2}) \ .$$

Proof. Since $\vec{x_1} \overset{X_2}{\approx} \vec{x_2}$, by Lemma 3.24, there are at least $n-p-|\mathcal{I}|$ equal entries in $\vec{x_1}$ and $\vec{x_2}$. Therefore, $\vec{x_2}$ can be considered a friend of $\vec{x_1}$ in the set $X_{(a'_i)_{i\in\mathcal{I}}\cup(x'_j)_{j\in\mathcal{J}}}$ (note that it is a class of vectors with *independent* components), where $\mathcal{J}$, such that $\mathcal{J} \cap \mathcal{I} = \emptyset$, is the set of the remaining p entries that are not necessarily equal and are not part of $\vec{a'}$. We can now apply Corollary 3.22, getting sensitivity $\sum_{i\in\mathcal{I}\cup\mathcal{J}} c_i = \sum_{i\in\mathcal{I}} c_i + \sum_{j\in\mathcal{J}} c_j$. In general, we do not know which $\mathcal{J}$ exactly will be taken for these particular $\vec{x}$ and $\vec{x'}$, but in any case $|\mathcal{J}| = \min(n-|\mathcal{I}|, p) \leq p$, and $p \cdot \max_{i\in[n]\setminus\mathcal{I}}(c_i)$ is a valid upper bound for $\sum_{j\in\mathcal{J}} c_j$. □

Corollary 3.27 (B-sensitivity w.r.t. component)**.** *Let mapping* $f : X \to Y$ *be* c_i *B-sensitive w.r.t. the component* ρ_i *for all* $i \in \{1, \ldots, n\}$*. Let* $\hat{\rho} := \bigsqcap_{i\in\mathcal{I}} \hat{\rho}_i$*. For all* $\hat{X}_1, \hat{X}_2 \in \hat{X}/\hat{\rho}$ *we have*

$$\forall \vec{x_1} \in \hat{X}_1, \vec{x_2} \in \hat{X}_2 : \ \vec{x_1} \overset{\hat{X}_2}{\approx} \vec{x_2} \wedge \vec{x_1} \overset{\hat{X}_2}{\sim} \vec{x_2} \implies d_Y(f(\vec{x_1}), f(\vec{x_2})) \leq \left(p \cdot \max_{i\in[n]\setminus\mathcal{I}} (c_i) + \sum_{i\in\mathcal{I}} c_i \right) \cdot \tilde{d}_X(\hat{X}_1, \hat{X}_2) \ .$$

Proof. As in the proof of Prop. 3.26, we have $\vec{x_1} \overset{X_{(a_i)_{i\in\mathcal{I}}\cup(x'_j)_{j\in\mathcal{J}}}}{\sim} \vec{x_2}$. By Prop. 3.20, any composite component $\bigsqcap_{i\in\mathcal{I}}\rho_i$ is equidistant, so $d_X(\vec{x},\vec{x'}) = \tilde{d}_X(X_{(a_i)_{i\in\mathcal{I}}\cup(x_j)_{j\in\mathcal{J}}}, X_{(a'_i)_{i\in\mathcal{I}}\cup(x'_j)_{j\in\mathcal{J}}})$, and we may apply Prop. 3.26, getting the upper bound $(p\cdot\max_{i\in[n]\setminus\mathcal{I}}(c_i)+\sum_{i\in\mathcal{I}}c_i)\cdot d_X(\vec{x_1},\vec{x_2})$. Now, by Prop. 3.25, each composite component $\hat{\rho} := \bigsqcap_{i\in\mathcal{I}}\hat{\rho}_i$ is also equidistant, and since $\vec{x_1} \overset{\hat{X}_2}{\sim} \vec{x_2}$, we have $d_X(\vec{x_1},\vec{x_2}) = \tilde{d}_X(\hat{X}_1,\hat{X}_2)$. □

We note that the conditions $\vec{x_1} \overset{\hat{X}_2}{\approx} \vec{x_2}$ and $\vec{x_1} \overset{\hat{X}_2}{\sim} \vec{x_2}$ are in general not simultaneously satisfiable. It may even happen than such $\vec{x_1}$ and $\vec{x_2}$ do not exist for some classes, since having similar coordinates may be harmful for reducing the distance due to the constraints.

For C- and D-sensitivities, we could use the assumption $\vec{X'_1} \overset{\hat{X}_2}{\approx} \vec{X'_2}$ instead of $\vec{x_1} \overset{\hat{X}_2}{\approx} \vec{x_2}$. However, when we reduce $\vec{X'_1} \overset{\hat{X}_2}{\approx} \vec{X'_2}$ directly to $\vec{X'_1} \overset{X'}{\sim} \vec{X'_2}$ for some set of vectors independent coordinates $X' \subseteq X$, it may happen that different pairs $x_1 \overset{\hat{X}_2}{\sim} x_2$ and $x'_1 \overset{\hat{X}_2}{\sim} x'_2$ have different similar components, and in the worst case we will only be able to take $X' = X$, which does not give any reasonable bounds. Adding restrictions on X'_1 and X'_2, that similar components of different matching pairs should be located on the same positions, would help. In this case, the condition $\vec{X'_1} \overset{\hat{X}_2}{\approx} \vec{X'_2} \wedge \vec{X'_1} \overset{\hat{X}_2}{\sim} \vec{X'_2}$ (needed for D-sensitivity) would be even more difficult to achieve.

3.3.5.6 Adding Restrictions on Constraints. In this section, we see which results we can achieve if we put some restrictions on $\mathbf{A}\vec{x} = \vec{b}$. Using Gaussian elimination, we may bring any underdetermined full-rank system to the form $\mathbf{A} = \mathbf{A'}|\mathbf{I}$, where $\mathbf{I}$ is an identity matrix, and $\mathbf{A'}$ is some matrix of $n-p$ columns. Since $\mathbf{I}$ does not necessarily have to be on the right, and can be scattered among any p columns, we may get different variants of $\mathbf{A'}$. Let us consider the case when it is possible to obtain $\|\mathbf{A'}\| < 1$, where $\|\mathbf{A'}\|$ is the *induced matrix norm*, i.e. $\|\mathbf{A'}\| = \sup_{\vec{x}\in\hat{X}}\frac{\|\mathbf{A'}\vec{x}\|}{\|\vec{x}\|}$.

In this section, we consider ℓ_1 norm and subsets of components indexed by $\mathcal{I}\subseteq[n-p]$. By properties of the induced ℓ_1 norm, $\|\mathbf{A}\| := \max_{j\in n}\sum_{i=1}^{p}|a_{ij}|$ for a $p\times n$ matrix $\mathbf{A}$. Note that the norm of a matrix does not change if we permute its columns in any way. Without loss of generality, we may assume that $\mathcal{I} = [m]$ for $m \le n-p$, since the columns of $\mathbf{A'}$ can be permuted arbitrarily without violating the condition $\|\mathbf{A'}\| < 1$.

Similarly to Sec. 3.3.5.5, we consider components of the form $\hat{\rho} := \bigsqcap_{i\in\mathcal{I}}\hat{\rho}_i$, where $\hat{\rho}_i$ is the analog of ρ_i, defined over $\hat{X}$. We use the same notation $\hat{X}_{(a_i)_{i\in[m]}}$ for equivalence classes.

Lemma 3.28. *Let $\mathcal{I} = [m]$, $m \le n-p$. Let $\hat{\rho} = \bigsqcap_{i\in[m]}\hat{\rho}_i$, and $\hat{X}_{(a_i)_{i\in[m]}}, \hat{X}_{(a'_i)_{i\in[m]}} \in \hat{X}/\hat{\rho}$. Let $\vec{x}\in\hat{X}$ be such that $x_i = a_i$ for $i\in[m]$, and $\mathbf{A}\vec{x} = \vec{b}$. Let $\vec{x'}\in\hat{X}$ be such that $\vec{x} \overset{\hat{X}_{(a'_i)_{i\in[m]}}}{\sim} \vec{x'}$, where distance is defined using ℓ_1 norm. Let $\mathbf{A} = \mathbf{A_1}|\mathbf{A_2}$, where $\mathbf{A_1}$ are the first m columns of $\mathbf{A}$.*

We can write $\vec{x'} = \vec{x}+\vec{z}$, where $\vec{z} = \vec{z_1}|\vec{z_2}$, $\vec{z_1} = \vec{a'}-\vec{a}$, and $\vec{z_2} = \vec{0}_{n-m-p}|\mathbf{A_1}(\vec{a}-\vec{a'})$.

Proof. From Lemma 3.23, we get that $\vec{x'} = \vec{x}+\vec{z}$, where $\vec{z} = \vec{z_1}|\vec{z_2}$, $\vec{z_1} = (\vec{a'}-\vec{a})$, and $\vec{z_2}$ is an optimal solution to the task

$$\text{minimize}\|\vec{z_2}\|,\ \text{subject to}\mathbf{A_2}\vec{z_2} = \mathbf{A_1}(\vec{a}-\vec{a'})\ . \tag{12}$$

By assumption, we have $\mathbf{A} = \mathbf{A'}|\mathbf{I}$, where $\mathbf{A'}$ are the first $n-p$ columns of $\mathbf{A'}$, such that $\|\mathbf{A'}\| < 1$. Hence, we have $\mathbf{A_2} = \mathbf{A''}|\mathbf{I}$, where $\mathbf{A''}$ are the remaining $n-m-p$ columns of $\mathbf{A'}$ (if any). By definition of the induced ℓ_1 norm, $\|\mathbf{A}\| := \max_{j\in n}\sum_{i=1}^{p}|a_{ij}|$ for a $p\times n$ matrix $\mathbf{A}$, so $\mathbf{A''} \le \mathbf{A'}$ since $\mathbf{A'}$ has more columns and, hence, more choices for the maximum. We propose that $\vec{z_2} = \vec{0}_{n-m-p}|\mathbf{A_1}(\vec{a}-\vec{a'})$, where $\vec{0}_{n-m-p}$ is a vector of $n-m-p$ zeroes, will be the optimal solution.

1. Substituting $\vec{z_2}$ with this value, we get $\mathbf{A_2}\vec{z_2} = \mathbf{A''}\cdot\vec{0}_{n-m-p} + \mathbf{I}\cdot\mathbf{A_1}(\vec{a}-\vec{a'}) = \mathbf{A_1}(\vec{a}-\vec{a'})$, so the constraints are satisfied. In this case, $\|z_2\| = \|\mathbf{A_1}(\vec{a}-\vec{a'})\|$.
2. Let now $\vec{z}$ be an arbitrary vector that satisfies the constraints. We have $\|\mathbf{A_1}(\vec{a}-\vec{a'})\| = \|\mathbf{A_2}\vec{z_2}\| = \|\mathbf{A''}z_{21} + \mathbf{I}z_{22}\|$. Using the triangle inequality and norm submultiplicativity, we get

$\|\mathbf{A}''\vec{z_{21}} + \mathbf{I}\vec{z_{22}}\| \leq \|\mathbf{A}''\vec{z_{21}}\| + \|\mathbf{I}\vec{z_{22}}\| \leq \|\mathbf{A}''\| \cdot \|\vec{z_{21}}\| + \|\vec{z_{22}}\| \leq \|\vec{z_{21}}\| + \|\vec{z_{22}}\|$. For ℓ_1 norm, $\|\vec{z_{21}}\| + \|\vec{z_{22}}\| = \|\vec{z_{21}}|\vec{z_{22}}\| = \|\vec{z}\|$.

Since we have a strict inequality $\|\mathbf{A}''\| < 1$, the last inequality in this chain is also strict whenever $\|\vec{z_{21}}\| \neq \vec{0}$, which holds for all $\vec{z_{21}} \neq \vec{0}$. In this way, any other solution is strictly worse than $\vec{z_2} = \vec{0}_{n-m-p}|\mathbf{A_1}(\vec{a} - \vec{a'})$. □

If we took $\|A'\| \leq 1$ instead of $\|A'\| < 1$ in Lemma 3.28, we would still get that $\vec{z_2} = \vec{0}_{n-m-p}|\mathbf{A}_{\{\mathbf{m+1}\}}(\vec{a} - \vec{a'})$ is an optimal solution, but there could possibly be some other optimal solutions that are of different form. If solutions of different form were allowed, we would not always be able construct a suitable mutual friend whose existence is required for the adjacent components.

Proposition 3.29. *Let $m \leq n - p - 1$. The components $\hat{\rho} := \bigsqcap_{i\in[m]} \hat{\rho}_i$ and $\hat{\sigma} := \hat{\rho}_{m+1}$ are adjacent.*

Proof. Let $\hat{X}_{(a_i)_{i\in[m]}}, \hat{X}_{(a')_{i\in[m]}} \in \hat{X}/\hat{\rho}$, and let $\hat{X}_a, \hat{X}_{a'} \in \hat{X}/\hat{\sigma}$. Let $\vec{x} = (a_1, \ldots, a_m, a, x_{m+2}, \ldots, x_n) \in \hat{X}_{(a)_{i\in[m]}} \cap \hat{X}_a$ and $\vec{x'} = (a'_1, \ldots, a'_m, a', x_{m+2}, \ldots, x_n) \in \hat{X}_{(a')_{i\in[m]}} \cap \hat{X}_{a'_{m+1}}$. By definition of adjacent components, their mutual friend should be some $\vec{x''} \in \hat{X}_{(a')_{i\in[m]}} \cap \hat{X}_{a_{m+1}}$. Let $\vec{a} := (a_1, \ldots, a_m)$, $\vec{a'} := (a'_1, \ldots, a'_m)$, $\vec{z} := \vec{x'} - \vec{x}$. Denote $\vec{a_1} := \mathbf{A}_{\{\mathbf{1},\ldots,\mathbf{m}\}}(\vec{a} - \vec{a'})$, $\vec{a_2} := \mathbf{A}_{\{\mathbf{m+1}\}}(a - a')$, and $\mathbf{A'} := \mathbf{A}_{\{\mathbf{m+2},\ldots,\mathbf{n}\}}$.

The first $m+1$ coordinates of $\vec{x''}$ are in any case equal to $(a'_1, \ldots, a'_m, a)$, and we cannot choose them. However, as far as $m + 1 \leq n - p$, it is still possible to find $x'' \in \hat{X}_{(a')_{i\in[m]}} \cap \hat{X}_{a_{m+1}}$ that does not violate the constraints (it is uniquely determined by constraints if $m + 1 = n - p$).

Let $m + 1 \leq n - p$. By Lemma 3.28, the vector $\vec{z}$ is such, that $\vec{z_2} = \vec{0}_{n-(m+1)-p}|\mathbf{A}_{\{\mathbf{1},\ldots,\mathbf{m+1}\}}(\vec{a}|(a) - \vec{a'}|(a')) = \vec{0}_{n-(m+1)-p}|(\vec{a_1} + \vec{a_2})$. Let us now take $\vec{x''} = \vec{x} + (\vec{y_1}|\vec{y_2})$, where $\vec{y_2} = \vec{0}_{n-(m+1)-p}|\vec{a_1}$. By Lemma 3.28, $\vec{x} \overset{\hat{X}_{(a')_{i\in[m]}}}{\sim} \vec{x''}$. On the other hand, $\|\vec{z_2} - \vec{y_2}\| = \vec{0}_{n-(m+1)-p}|(\vec{a_1} + \vec{a_2} - \vec{a_1}) = \vec{0}_{n-(m+1)-p}|\vec{a_2}$. Hence, by Lemma 3.28, also $\vec{x'} \overset{\hat{X}_{a_{m+1}}}{\sim} \vec{x''}$. □

Proposition 3.30. *The components $\hat{\rho} := \bigsqcap_{i\in[m]} \hat{\rho}_i$ and $\hat{\sigma} := \hat{\rho}_{m+1}$ are in general NOT expanding.*

Proof. Let $\hat{X}_{(a_i)_{i\in[m]}}, \hat{X}_{(a'_i)_{i\in[m]}} \in \hat{X}/\hat{\rho}$, and let $\hat{X}_a, \hat{X}_{a'} \in \hat{X}/\hat{\sigma}$. By Lemma 3.28, the distance between all friends of the sets $\hat{X}_{(a_i)_{i\in[m]}}$ and $\hat{X}_{(a'_i)_{i\in[m]}}$ is $\|\mathbf{A}_{\{\mathbf{1},\ldots,\mathbf{m}\}}(\vec{a} - \vec{a'})\|$. For the finer component $\hat{\rho} \sqcap \hat{\sigma}$, this distance is $\|\mathbf{A}_{\{\mathbf{m+1}\}}(a - a')\|$. In general, we do not know whether it is larger or smaller. It depends not only on the constraints, but also on the particular classes that we consider, so we also cannot give sufficient conditions for $\mathbf{A}$ that would make all components expanding. □

Similarly to vectors with independent components, the adjacent property allows us to apply A- and C-sensitivity using Theorem 3.13 and Theorem 3.16. Although the components are not expanding, since they are equidistant, we can also apply B- or D-sensitivity using Corollary 3.14 and Corollary 3.17. However, the components are adjacent only as far as we compose up to $n-p$ of them. After that, we may no longer use the common friend since the corresponding component may be empty due to the violation of constraints, and then we may only find the bounds as in Sec. 3.3.5.5.

3.3.5.7 Sensitivity of Function Composition. We want to know how sensitivity w.r.t. components depends on function composition. Given functions $f : X \to Y$ and $g : Y \to Z$, and a component ρ of X and a component σ of Y, it may be too difficult to estimate the sensitivity of $g \circ f$ w.r.t. ρ if we know only sensitivity of g w.r.t. σ, since f may map elements of X to very different classes of σ. If x and x' belong to different ρ-classes, it does not mean that $f(x)$ and $f(x')$ belong to different σ classes, unless we make σ dependent on ρ and f. Even in this case, we would require that f is injective to avoid overlapping of σ classes. If we do not put any additional constraints on f, we will need to assume that we know the general sensitivity of g (not w.r.t. some component).

Theorem 3.31. *Let $f : X \to Y$ be c_f sensitive w.r.t. component ρ. Let $g : Y \to Z$ be c_g sensitive. Then, the function composition $g \circ f$ is $c_f \cdot c_g$ sensitive w.r.t. ρ, using any definition of sensitivity (A-,B-,C-,D-).*

Proof. We give the proof for different types of sensitivity of f. Let $X_1, X_2 \in X/\rho$.

1. Let $x \in X_1$, $x' \in X_2$, $x \overset{X_2}{\sim} x'$. If f is A-sensitive w.r.t. ρ, we have $d_Y(f(x), f(x')) \leq c_f \cdot d_X(x, x')$. Since g is c_g sensitive, we have $d_Z(g(y), g(y')) \leq c_g \cdot d_Y(y, y')$ for all $y, y' \in Y$, and hence also for $y = f(x)$, $y' = f(x')$. Putting the two inequalities together, we get $d_Z(gf(x), gf(x')) \leq c_g \cdot d_Y(f(x), f(x')) \leq c_g \cdot c_f \cdot d_X(x, x')$.
2. The proof is similar for B-sensitivity, using $\tilde{d}_X(X_1, X_2)$ instead of $d_X(x, x')$.
3. If $\tilde{d}_Y(\cdot, \cdot)$ is Hausdorff distance, then by Theorem 3.12, C-sensitivity is equivalent to A-sensitivity, and the composition is reduced to composition of A-sensitivity. Let $\tilde{d}_Y(\cdot, \cdot)$ be max-distance. Let $X_1' \subseteq X_1$, $X_2' \subseteq X_2$ be two friendly sets. Denote $Y_1 := f(X_1')$, $Y_2 := f(X_2')$. Let $y_1 \in Y_1$ and $y_2 \in Y_2$ be such that $g(y_1) \overset{g(Y_2)}{\sim} g(y_2)$ and $\tilde{d}_Z(g(Y_1), g(Y_2)) = d_Z(g(y_1), g(y_2))$. Since g is c_g sensitive, we have $\tilde{d}_Z(g(Y_1), g(Y_2)) = d_Z(g(y_1), g(y_2)) \leq c_g \cdot d_Y(y_1, y_2)$.
   Since $\tilde{d}_Y(\cdot, \cdot)$ is max-distance, $d_Y(y_1, y_2) \leq \tilde{d}_Y(Y_1, Y_2)$. Putting it together with $\tilde{d}_Y(f(X_1'), f(X_2')) \leq c_f \cdot \tilde{d}_X(X_1', X_2')$, we get $\tilde{d}_Z(gf(X_1'), gf(X_2')) \leq c_g \cdot c_f \cdot \tilde{d}_X(X_1', X_2')$.
4. The proof is similar for D-sensitivity, using $\tilde{d}_X(X_1, X_2)$ instead of $\tilde{d}_X(X_1', X_2')$. □

3.3.6 Categorical View of Sensitivity and Distances. Sec. 3.3.5 shows the need to talk about sensitivities of multivariate mappings with respect to a certain argument. We can foresee the need to argue about sensitivities with respect to certain combinations of inputs, and even not knowing precisely which sensitivities are needed in compositions. To keep track of the sensitivities of the workflow components with respect to their different inputs and outputs, we have to add more structure to both distances and sensitivities themselves.

3.3.6.1 Generalized Distances and Sensitivities.

Definition 3.28 (Set of distances)**.** A set V with the following additional algebraic structure is a suitable *set of distances*:

- V is a *commutative monoid*, i.e. there is an operation $+ : V \times V \to V$ and an element $0 \in V$, such that
  - $(u + v) + w = u + (v + w)$ for all $u, v, w \in V$;
  - $u + v = v + u$ for all $u, v \in V$;
  - $u + 0 = u$ for all $u \in V$.
- V is a *partial order*, i.e. there is a relation $\leq \subseteq V \times V$, such that
  - $u \leq u$ for all $u \in V$;
  - if $u \leq v$ and $v \leq u$ then $u = v$ for all $u, v \in V$;
  - $u \leq v$ and $v \leq w$ imply $u \leq w$ for all $u, v, w \in V$.
- Addition and order are compatible in the sense that 0 is the least element and all other elements may be considered "positive". Formally,
  - $0 \leq u$ for all $u \in V$;
  - If $u \leq v$ then $u + w \leq v + w$ for all $u, v, w \in V$.

Obviously, $\mathbb{R}_+$ with addition and ordering satisfies this definition. For any n, the set of n-tuples $\mathbb{R}_+^n$ also satisfies it, if both addition and ordering are defined componentwise. We expect to such distances with several components to have a number of uses. We use the set of distances for generalizing the notion of a metric space.

Definition 3.29 (Generalized metric space)**.** A *generalized metric space* is a triple (X, d_X, V_X), where X is a set, V_X is a set of distances, and the *metric* $d_X : X \times X \to V_X$ satisfies

- for all $x, y \in X$, $d_X(x, y) = 0$ iff $x = y$;

- for all $x, y \in X$, $d_X(x, y) = d_X(y, x)$;
- for all $x, y, z \in Z$, $d_X(x, z) \leq d_X(x, y) + d_X(y, z)$.

We see that this definition coincides with Def. 3.1, except that we are using an arbitrary set of distances V_X instead of $\mathbb{R}_+$. The sensitivities of mappings are also generalized. In the following, let $V_X \xrightarrow{\text{m}} V_Y$ denote the set of *order-preserving* functions from V_X to V_Y.

Definition 3.30 (Generalized sensitivity)**.** Let (X, d_X, V_X) and (Y, d_Y, V_Y) be generalized metric spaces. Let $f : X \to Y$ and $c : V_X \xrightarrow{\text{m}} V_Y$. We say that the *sensitivity* of f is (at most) c if for all $x, x' \in X$, $d_Y(f(x), f(x')) \leq c(d_X(x, x'))$.

Def. 3.30 indeed generalizes Def. 3.5. Let $c_0 \in \mathbb{R}_+$ be the (ordinary) sensitivity of some mapping $f : X \to Y$ between (ordinary) metric spaces X and Y with distances d_X and d_Y. If we consider them as generalized metric spaces $(X, d_X, \mathbb{R}_+)$ and $(Y, d_Y, \mathbb{R}_+)$, then the sensitivity of f is $c : \mathbb{R}_+ \to \mathbb{R}_+$, where $c(v) = c_0 \cdot v$ for all $v \in \mathbb{R}_+$.

Proposition 3.1 can be generalized:

Proposition 3.32. *Let (X, d_X, V_X), (Y, d_Y, V_Y) and (Z, d_Z, V_Z) be generalized metric spaces. Let $f : X \to Y$ have the sensitivity c and $f' : Y \to Z$ the sensitivity c'. Then Then $f' \circ f : X \to Z$ has sensitivity $c' \circ c$.*

Proof. Let $x, x' \in X$. Then

$$d_Z(f'(f(x)), f'(f(x'))) \leq c'(d_Y(f(x), f(x'))) \leq c'(c(d_X(x, x'))) \ . \qquad \square$$

In the next section we will see that the generalizations of both Prop. 3.3 and Prop. 3.4 are easy consequences of Prop. 3.32.

For the rest of this section, we make the following notational convention. We will denote generalized metric spaces with capital letters: X, Y, Z, etc. For a generalized metric space X, we let $\mathrm{Carr}(X)$ denote the "carrier set", i.e. the first component of X, $\mathrm{SoD}(X)$ the set of distances used in X, and d_X the metric, mapping pairs of elements of $\mathrm{Carr}(X)$ to $\mathrm{SoD}(X)$.

Before moving on to constructions of generalized metric spaces with more structure, let us mention a quite trivial one. Let X and X' be generalized metric spaces, such that $\mathrm{Carr}(X) = \mathrm{Carr}(X')$. Let X'' be defined by

- $\mathrm{Carr}(X'') = \mathrm{Carr}(X)$;
- $\mathrm{SoD}(X'') = \mathrm{SoD}(X) \times \mathrm{SoD}(X')$, where addition and order are defined componentwise;
- $d_{X''}(x_1, x_2) = (d_X(x_1, x_2), d_{X'}(x_1, x_2))$.

Then X'' is also a generalized metric space. Its metric captures the information present in the metrics of both X and X'.

Generalized distances and metric spaces have been studied in the past, even though the goals have been different. It turns out that any topology over a set arises from a generalized metric over it [51]. This is not the case for ordinary metrics.

3.3.6.2 Limits of Generalized Metric Spaces. Limits in *categories* are certain unique (up to isomorphism) objects, through which certain arrows factorize. They generalize products, as well as inductive datatypes. We start with perhaps the simplest construction of a more complex generalized metric space from simpler ones.

Proposition 3.33. *Let X and Y be generalized metric spaces. Then*

$$X \times Y := (\mathrm{Carr}(X) \times \mathrm{Carr}(Y), d_{X \times Y}, \mathrm{SoD}(X) \times \mathrm{SoD}(Y))$$

is also a generalized metric space, where the ordering and addition on $\mathrm{SoD}(X) \times \mathrm{SoD}(Y)$ is defined componentwise, and $d_{X \times Y}((x_1, y_1), (x_2, y_2)) = (d_X(x_1, x_2), d_Y(y_1, y_2))$ for all $x_1, x_2 \in \mathrm{Carr}(X)$ and $y_1, y_2 \in \mathrm{Carr}(Y)$.

Proof. We have to verify that $\mathrm{SoD}(X) \times \mathrm{SoD}(Y)$ is a set of distances and $d_{X\times Y}$ is a metric. Both checks are trivial, following directly from the definitions. □

This simple way of building a product generalized metric space would not have been available if we had considered only ordinary metrics. Proposition 3.33 presents the "correct" way of defining products in the following sense. Consider the category **GM**, defined as follows:

- The objects of **GM** are generalized metric spaces.
- An arrow f from a generalized metric space X to a generalized metric space Y is a pair (k, c), where $k : \mathrm{Carr}(X) \to \mathrm{Carr}(Y)$, $c : \mathrm{SoD}(X) \xrightarrow{\mathrm{m}} \mathrm{SoD}(Y)$, and k is c-sensitive. We denote the components of an arrow $f = (k, c)$ by $\mathrm{f}[f]$ and $\mathrm{c}[f]$.

For each object, there exists an identity arrow: the identity map together with identity sensitivity. The composition of arrows is defined componentwise; according to Prop. 3.32, this again gives us an arrow. The category axioms are obviously satisfied.

Proposition 3.34. *The category* **GM** *has products, they generalize the binary product construction given in Prop. 3.33. In more details: for an index set I, and a set of generalized metric spaces $(X_i)_{i\in I}$, there exists a generalized metric space Y, together with arrows $\pi_i : Y \to X_i$ for all $i \in I$, such that for any generalized metric space Z together with arrows $f_i : Z \to X_i$ there exists exactly one arrow $g : Z \to Y$, such that $f_i = \pi_i \circ g$.*

Proof. Let $\mathrm{Carr}(Y) = \prod_{i\in I} \mathrm{Carr}(X_i)$, $\mathrm{SoD}(Y) = \prod_{i\in I} \mathrm{SoD}(X_i)$ and $d_Y((x_i)_{i\in I}, (x'_i)_{i\in I}) = (d_{X_i}(x_i, x'_i))_{i\in I}$, where $\prod$ denotes the Cartesian product of sets. Define $\mathrm{f}[\pi_j]((x_i)_{i\in I}) = x_j$ and $\mathrm{c}[\pi_j]((v_i)_{i\in I}) = v_j$, where $x_i \in \mathrm{Carr}(X_i)$ and $v_i \in \mathrm{SoD}(X_i)$.

For the given generalized metric space Z and arrows f_i, define $\mathrm{f}[g](z) = (\mathrm{f}[f_i](z))_{i\in I} \in \mathrm{Carr}(Y)$ and $\mathrm{c}[g](v) = (\mathrm{c}[f_i](v))_{i\in I} \in \mathrm{SoD}(Y)$. Clearly, $f_i = \pi_i \circ g$. For any other g', some of these equalities must not hold. Indeed, if there is some $j \in I$, such that for some $z \in \mathrm{Carr}(Z)$ we have $\mathrm{f}[g](z)$ and $\mathrm{f}[g'](z)$ differing in j-th component, or for some $v \in \mathrm{SoD}(Z)$ we have $\mathrm{c}[g](v)$ and $\mathrm{c}[g'](v)$ differing in the j-th component, then $f_j \neq \pi_j \circ g'$. □

Typically, Y is denoted by $\prod_{i\in I} X_i$ and g by $\langle f_i\rangle_{i\in I}$. For arrows $f : X \to Y$ and $f' : X' \to Y'$, one denotes $f \times f' := \langle f \circ \pi_1, f' \circ \pi_2\rangle : X \times X' \to Y \times Y'$, where $\pi_1 : X \times X' \to X$ and $\pi_2 : X \times X' \to X'$ are given by the product construction.

We can now state the generalizations of Prop. 3.3 and Prop. 3.4.

Proposition 3.35. *Let X, Y, Z, W be generalized metric spaces. Consider the arrows $f : X \to Y$, $a : Y \to Z$, $g : X \times Z \to W$. Then there exists an arrow $h : X \to W$ with $\mathrm{f}[h](x) = \mathrm{f}[g](x, \mathrm{f}[a](\mathrm{f}[f](x))$ for all $x \in \mathrm{Carr}(X)$ and $\mathrm{c}[h](v) = \mathrm{c}[g](v, \mathrm{c}[a](\mathrm{c}[f](v)))$ for all $v \in \mathrm{SoD}(X)$.*

Proof. Take $h = g \circ (id_X \times (a \circ f))$. □

Prop. 3.35 is specialized to Prop. 3.3 by taking $\mathrm{SoD}(X) = \mathrm{SoD}(Y) = \mathrm{SoD}(Z) = \mathrm{SoD}(W) = \mathbb{R}_+$, $\mathrm{c}[f](v) = c \cdot v$, $\mathrm{c}[a](v) = v$, and $\mathrm{c}[g](v, v') = c' \cdot v + v'$ for all $v, v' \in \mathbb{R}_+$.

Prop. 3.4 does not simplify that much when going to generalized metric spaces. Nevertheless, we can state it as follows. Given a partially ordered set V and a subset $V' \subseteq V$, we let $\mathrm{UB}\{V'\} \subseteq V$ denote the set of all upper bounds of V': elements that are greater or equal to all elements of V'. A mapping c between sets of distances is *superadditive* if $c(v_1 + v_2) \geq c(v_1) + c(v_2)$ for all v_1, v_2 in the domain of c.

Proposition 3.36. *Let X, Y, Z, W, f, a, g be as in Prop. 3.35. Let ρ and σ be equivalence relations on X that are independent. Let $x\ \rho\ x'$ imply $\mathrm{f}[f](x) = \mathrm{f}[f](x')$, and $x\ \sigma\ x'$ imply $\mathrm{f}[g](x, z) = \mathrm{f}[g](x', z)$ for all $x, x' \in \mathrm{Carr}(X)$ and $z \in \mathrm{Carr}(Z)$. Let $c : \mathrm{SoD}(X) \xrightarrow{\mathrm{m}} \mathrm{SoD}(W)$ be any mapping that satisfies the following conditions:*

- *for all $v \in \mathrm{SoD}(X)$, $c(v) \in \mathrm{UB}\{\mathrm{c}[g](v, 0), \mathrm{c}[g](0, \mathrm{c}[a](\mathrm{c}[f](v)))\}$;*
- *c is superadditive.*

Then there exists an arrow $h : X \to W$ *with* $\mathrm{f}[h](x) = \mathrm{f}[g](x, \mathrm{f}[a](\mathrm{f}[f](x))$ *for all* $x \in \mathrm{Carr}(X)$ *and* $\mathrm{c}[h] = c$.

Proof. Let $x, x' \in \mathrm{Carr}(X)$. Let m and $x_0, \ldots, x_{2m}$ be given by the definition of independence of ρ and σ. Similarly to the proof of Prop. 3.4, we get

$$d_W(\mathrm{f}[h](x), \mathrm{f}[h](x')) \leq \left(\sum_{i=1}^{m} d_W(\mathrm{f}[g](x_{2i-2}, \mathrm{f}[a](\mathrm{f}[f](x_{2i-2}))), \mathrm{f}[g](x_{2i-1}, \mathrm{f}[a](\mathrm{f}[f](x_{2i-2}))))\right) + \left(\sum_{i=1}^{m} d_W(\mathrm{f}[g](x_{2i}, \mathrm{f}[a](\mathrm{f}[f](x_{2i-1}))), \mathrm{f}[g](x_{2i}, \mathrm{f}[a](\mathrm{f}[f](x_{2i}))))\right) .$$

Let $v_i = d_X(x_{i-1}, x_i)$. We have

$$d_W(\mathrm{f}[g](x_{2i-2}, \mathrm{f}[a](\mathrm{f}[f](x_{2i-2}))), \mathrm{f}[g](x_{2i-1}, \mathrm{f}[a](\mathrm{f}[f](x_{2i-2})))) \leq \mathrm{c}[g](v_{2i-1}, 0) \leq c(v_{2i-1})$$
$$d_W(\mathrm{f}[g](x_{2i}, \mathrm{f}[a](\mathrm{f}[f](x_{2i-1}))), \mathrm{f}[g](x_{2i}, \mathrm{f}[a](\mathrm{f}[f](x_{2i})))) \leq \mathrm{c}[g](0, \mathrm{c}[a](\mathrm{c}[f](v_{2i}))) \leq c(v_{2i})$$

By superadditivity of c,

$$d_W(\mathrm{f}[h](x), \mathrm{f}[h](x')) \leq \sum_{i=1}^{2m} c(v_i) \leq c(\sum_{i=1}^{2m} v_i) = c(d_X(x, x')) \ . \qquad \square$$

Products are one kind of limits. In order to have canonical methods for constructing complex data structures, we are interested also in the existence of other limits. In particular, we are interested in the existence of *equalizers*, because they, together with products, allow to construct all other limits. But as we see below, **GM** does not have equalizers.

Definition 3.31 (Equalizer)**.** Let $\mathbf{C}$ be a category, X, Y two objects in it, and $f_1, f_2 : X \to Y$ two arrows between these objects. An *equalizer* of f_1, f_2 is an object Z in $\mathbf{C}$, together with an arrow $g : Z \to X$, such that $f_1 \circ g = f_2 \circ g$. Moreover, for any object Z' and arrow $g' : Z' \to X$, if $f_1 \circ g' = f_2 \circ g'$, then there exists a unique arrow $h : Z' \to Z$, such that $g' = g \circ h$.

Consider now the category **GM**, two generalized metric spaces X and Y, and two arrows f_1, f_2 between them. An equalizer $g : Z \to X$ has to "make equal" both $\mathrm{f}[f_1]$ and $\mathrm{f}[f_2]$, as well as $\mathrm{c}[f_1]$ and $\mathrm{c}[f_2]$. It also has to be the largest among those that "make them equal".

In the category of sets, an equalizer is the subset of X where f_1 and f_2 agree. In **GM**, we must have agreement in both $\mathrm{Carr}(X)$ and $\mathrm{SoD}(X)$. Hence we must have

$$\mathrm{Carr}(Z) \subseteq \{x \in \mathrm{Carr}(X) \mid \mathrm{f}[f_1](x) = \mathrm{f}[f_2](x)\}$$
$$\mathrm{SoD}(Z) \subseteq \{v \in \mathrm{SoD}(X) \mid \mathrm{c}[f_1](v) = \mathrm{c}[f_2](v)\} \ .$$

Also, $\mathrm{SoD}(Z)$ has to be closed with respect to addition.

The following example demonstrates two arrows (with the same domain and codomain) in **GM** that have no equalizer.

Example 3.4. Consider the following generalized metric spaces X and Y, as well as arrows $f_1, f_2 : X \to Y$:

- $\mathrm{Carr}(X) = \{1_1, 1_2, 2, 3\}$ and $\mathrm{Carr}(Y) = \{1, 2, 3\}$;
- $\mathrm{SoD}(X) = \{0, 1, 2, 3\}$, where ordering is defined by $0 \leq 1 \leq 2 \leq 3$ and addition is defined by $0 + v = v$, and $u + v = 3$ if $u \neq 0 \neq v$;
- $\mathrm{SoD}(Y) = \{0, A, B, C_1, C_2, D\}$, where $0 \leq A \leq B \leq C_1 \leq D$, $B \leq C_2 \leq D$ and the addition is given by the table below.

- d_X and d_Y are given by the tables below.
- $\mathrm{f}[f_1] = \mathrm{f}[f_2] = \{1_1 \mapsto 1, 1_2 \mapsto 1, 2 \mapsto 2, 3 \mapsto 3\}$
- $\mathrm{c}[f_1]$ and $\mathrm{c}[f_2]$ are given in the tables below.

$+$	A	B	C_1	C_2	D
A	B	D	D	D	D
B	D	D	D	D	D
C_1	D	D	D	D	D
C_2	D	D	D	D	D
D	D	D	D	D	D

d_X	1_1	1_2	2	3
1_1	0	1	1	2
1_2	1	0	1	2
2	1	1	0	1
3	2	2	1	0

v	0	1	2	3
$\mathrm{c}[f_1](v)$	0	A	C_1	D

d_Y	1	2	3
1	0	A	B
2	A	0	A
3	B	A	0

v	0	1	2	3
$\mathrm{c}[f_2](v)$	0	A	C_2	D

If Z and g were an equalizer for f_1, f_2, then $\mathrm{Carr}(Z) \subseteq \{1_1, 1_2, 2, 3\}$ and $\mathrm{SoD}(Z) \subseteq \{0, 1, 3\}$, while both $\mathrm{f}[g]$ and $\mathrm{c}[g]$ are identity mappings. Taking $\mathrm{Carr}(Z) = \{1_1, 1_2, 2\}$ and $\mathrm{SoD}(Z) = \{0, 1, 3\}$ would "make f_1 and f_2 equal". Alternatively, taking $\mathrm{Carr}(Z) = \{2, 3\}$ and $\mathrm{SoD}(Z) = \{0, 1, 3\}$ would "make f_1 and f_2 equal". But these two options are not isomorphic and there is no option that is larger than both of them.

3.3.6.3 Colimits of Generalized Metric Spaces. Sums are dual for products, and necessary for defining more complex algebraic data structures. A natural definition for binary sums is given by the following proposition.

Proposition 3.37. *Let X and Y be generalized metric spaces. Define Z as follows:*

- $\mathrm{Carr}(Z) = \mathrm{Carr}(X) \uplus \mathrm{Carr}(Y)$*, where* $\uplus$ *denotes the* disjoint union*, typically defined as* $A \uplus B = (\{0\} \times A) \cup (\{1\} \times B)$*.*
- $\mathrm{SoD}(Z) = \{0, \infty\} \uplus (\mathrm{SoD}(X)\backslash\{0\}) \uplus (\mathrm{SoD}(Y)\backslash\{0\})$*, with the order and addition defined as follows:*
  - *0 is the smallest and ∞ the largest element. For $v_1, v_2 \in \mathrm{SoD}(X)\backslash\{0\}$ [resp. $\mathrm{SoD}(Y)\backslash\{0\}$], the ordering is the same as in* $\mathrm{SoD}(X)$ *[resp.* $\mathrm{SoD}(Y)$*]. If $v_1 \in \mathrm{SoD}(X)\backslash\{0\}$ and $v_2 \in \mathrm{SoD}(Y)\backslash\{0\}$, then v_1 and v_2 are incomparable.*
  - *$v + 0 = v$ and $v + \infty = \infty$ for all $v \in \mathrm{SoD}(Z)$. If v_1 and v_2 both belong to either* $\mathrm{SoD}(X)$ *or* $\mathrm{SoD}(Y)$*, then they are added as in this set. If $v_1 \in \mathrm{SoD}(X)\backslash\{0\}$ and $v_2 \in \mathrm{SoD}(Y)\backslash\{0\}$, then $v_1 + v_2 = \infty$.*
- *For $x_1, x_2 \in \mathrm{Carr}(X)$, $d_Z(x_1, x_2) = d_X(x_1, x_2)$. Similarly, $d_Z(y_1, y_2) = d_Y(y_1, y_2)$ for $y_1, y_2 \in \mathrm{Carr}(Y)$. If $x \in \mathrm{Carr}(X)$ and $y \in \mathrm{Carr}(Y)$, then $d_Z(x, y) = \infty$.*

Then Z is a generalized metric space.

Proof. Similarly to the proof of Prop. 3.33, we have to check that $\mathrm{SoD}(Z)$ is a set of distances and d_Z is a metric. Both checks are trivial. □

We denote Z by $X + Y$. Unfortunately, $X + Y$ is not the coproduct of X and Y in the category **GM**. For (Z, ι_1, ι_2), where $\iota_1 : X \to Z$ and $\iota_2 : Y \to Z$, to be the coproduct of X and Y, there must exist a unique arrow $g : Z \to W$ for any generalized metric space W and arrows $f_1 : X \to W$ and $f_2 : Y \to W$, such that $f_1 = g \circ \iota_1$ and $f_2 = g \circ \iota_2$. Our construction does not fix the value of $\mathrm{c}[g](\infty)$ in any way, or even guarantee its existence. Hence, depending on (W, f_1, f_2), the arrow g might not exist at all, or it might not be unique.

We can get back the coproducts if we put extra restrictions on distances and sensitivities. Namely, let $\mathbf{GM}_0^\infty$ be a subcategory of **GM** containing only such objects and arrows, where

- the sets of distances contain the largest element (denoted ∞);
- the sensitivities preserve both 0 and ∞.

Note that $\mathbf{GM}_0^\infty$ is closed with respect to products. In $\mathbf{GM}_0^\infty$, we can define the set of distances of the sum as the disjoint union of the sets of distances of the components, where we do not duplicate the distances 0 and ∞.

Having sums and products, we can define more complex data structures in $\mathbf{GM}_0^\infty$, automatically obtaining a canonical set of distances for it. E.g. for a generalized metric space X we can define the generalized metric space $\mathrm{List}(X)$ with the following underlying set and the following set of distances:

- $\mathrm{Carr}(\mathrm{List}(X)) = \mathrm{Carr}(X)^*$;
- $\mathrm{SoD}(\mathrm{List}(X)) = \{0, \infty\} \uplus (\mathrm{SoD}(X)^* \backslash (0^* \cup \infty^*))$.

However, the resulting distance $d_{\mathrm{List}(X)}$ is not particularly interesting. If two lists of elements of $\mathrm{Carr}(X)$ have the same length n, then their distance is defined componentwise, being an element of $\mathrm{SoD}(X)^n$ (with 0^n being identified with the least element 0). If the lists have different lengths, then their distance is ∞.

We saw that the category **GM** had no equalizers. Interestingly, it has coequalizers. The subcategory $\mathbf{GM}_0^\infty$ is closed with respect to the construction we give below, meaning that $\mathbf{GM}_0^\infty$ has both coproducts and colimits. Hence $\mathbf{GM}_0^\infty$ is *cocomplete* — it has all colimits. It remains to be seen what implications this has with respect to the existence of natural definitions of different kinds of algebraic data structures.

Definition 3.32 (Coequalizer)**.** Let $\mathbf{C}$ be a category, X, Y two objects in it, and $f_1, f_2 : X \to Y$ two arrows between these objects. A *coequalizer* of f_1, f_2 is an object Z in $\mathbf{C}$, together with an arrow $g : Y \to Z$, such that $g \circ f_1 = g \circ f_2$. Moreover, for any object Z' and arrow $g' : Y \to Z'$, if $g' \circ f_1 = g' \circ f_2$, then there exists a unique arrow $h : Z \to Z'$, such that $g' = h \circ g$.

In the category of sets, the equalizer is a factor set of Y, namely $Z = Y/\rho$, where ρ is the finest equivalence relation that relates $f(x), g(x)$ for all $x \in X$. In the category **GM**, a coequalizer needs to make equal not only f[f_1] and f[f_2], but also c[f_1] and c[f_2]. To show how this is possible, let us discuss congruences (i.e. structure-preserving equivalences) on sets of distances.

Definition 3.33 (Congruence on a set of distances)**.** Let V be a set of distances. We say that $\sigma \subseteq V \times V$ is a congruence on V, if

- σ is an equivalence relation, i.e. it is reflexive, symmetric and transitive;
- σ is a congruence on the commutative monoid $(V, +, 0)$, i.e. $x \,\sigma\, y$ and $w \,\sigma\, z$ imply $x + w \,\sigma\, y + z$;
- σ preserves ordering, meaning that for any $v_0, v_1, \ldots, v_{2m}$ satisfying

$$v_0 \le v_1 \,\sigma\, v_2 \le v_3 \,\sigma\, v_4 \le \cdots \,\sigma\, v_{2m} = v_0, \tag{13}$$

  the relations $v_i \,\sigma\, v_j$ hold for any $i, j \in \{0, \ldots, 2m\}$.

Proposition 3.38. *Let V be a set of distances and σ a congruence on it. Let $V/\sigma = \{[v]_\sigma \mid v \in V\}$, where $[v]_\sigma = \{u \in V \mid u \,\sigma\, v\}$. Define the following algebraic structure on V/σ:*

- *Addition: $[u]_\sigma + [v]_\sigma = [u + v]_\sigma$;*
- *Ordering: $[u]_\sigma \le [v]_\sigma$ iff there exist $u_0, v_0, \ldots, u_m, v_m \in V$, such that*

$$u \,\sigma\, u_0 \le v_0 \,\sigma\, u_1 \le v_1 \,\sigma\, u_2 \le v_2 \,\sigma \cdots \le v_m \,\sigma\, v; \tag{14}$$

Then V/σ is a set of distances.

Proof. We have to verify a number of things. First: V/σ is a commutative monoid. This follows from σ being a congruence on $(V, +, 0)$.

Second, V/σ is a partially ordered set. Reflexivity of $\le$ is immediate. Transitivity is an easy consequence of (14). Consider whether $\le$ is antisymmetric. Suppose $[u]_\sigma \le [v]_\sigma$ and $[v]_\sigma \le [u]_\sigma$. By the definition of ordering, there exist $u_0, v_0, \ldots, u_m, v_m$ and $v'_0, u'_0, \ldots, v'_n, u'_n$, such that

$$u \,\sigma\, u_0 \le v_0 \,\sigma\, u_1 \le v_1 \,\sigma \cdots \le v_m \,\sigma\, v \,\sigma\, v'_1 \le u'_1 \,\sigma\, v'_2 \le u'_2 \,\sigma \cdots \le u'_n \,\sigma\, u \ .$$

By the order-preservation of σ we obtain $u \sigma v$ and hence $[u]_\sigma = [v]_\sigma$.

Third, addition and ordering are compatible. We have $[0]_\sigma \leq [v]_\sigma$ because $0 \leq v$. Suppose $[u]_\sigma \leq [v]_\sigma$ and $w \in V$. The relation (14), together with σ being a monoid congruence and the compatibility of addition and ordering on V implies

$$u + w \sigma u_0 + w \leq v_0 + w \sigma u_1 + w \leq v_1 + w \sigma u_2 + w \leq v_2 + w \sigma \cdots \leq v_m + w \sigma v + w,$$

hence $[u + w]_\sigma \leq [v + w]_\sigma$. □

For equivalences over sets, there exists the smallest equivalence relation that relates certain pairs of values. It turns out that for congruences over distances, the situation is similar. Namely, if σ_i are congruences on a set of distances V, where $i \in I$ for some index set I, then $\sigma = \bigcap_{i \in I} \sigma_i$ is also a congruence on V. Clearly, σ is an equivalence relation and a congruence on the commutative monoid $(V, +, 0)$. Also, if σ preserves ordering, i.e. (13) holds for σ, then it also holds for each σ_i, hence all v_j-s are related by each σ_i, hence they are also related by σ. Now, given pairs $\{(u_j, v_j)\}_{j \in J}$, the smallest congruence σ on V that satisfies $u_j \sigma v_j$ for each $j \in J$, is simply the intersection of all congruences that satisfy these requirements.

In the following, given $U \subseteq V \times V$, let $\operatorname{cl} U$ denote the smallest equivalence relation (if V is an arbitrary set) or the smallest congruence (if V is a set of distances) that contains U. We can now describe the coequalizers.

Proposition 3.39. *The categories* **GM** *and* $\mathbf{GM}_0^\infty$ *have coequalizers.*

Proof. Let X, Y be two generalized metric spaces and $f_1, f_2 : X \to Y$ two arrows. Define the equivalence relations ρ over $\operatorname{Carr}(Y)$ and σ over $\operatorname{SoD}(Y)$ as follows:

$$\begin{aligned}
\rho &= \operatorname{cl}\{(\mathrm{f}[f_1](x), \mathrm{f}[f_2](x)) \mid x \in \operatorname{Carr}(X)\} \\
\sigma &= \operatorname{cl}\big(\{(\mathrm{c}[f_1](u), \mathrm{c}[f_2](u)) \mid u \in \operatorname{SoD}(X)\} \cup \{(d_Y(y_1, y_2), d_Y(y_1', y_2')) \mid y_1 \rho y_1', y_2 \rho y_2'\}\big) \ . \qquad (15)
\end{aligned}$$

Construct a generalized metric space Z as follows:

$$\begin{aligned}
\operatorname{Carr}(Z) &= \operatorname{Carr}(Y)/\rho \\
\operatorname{SoD}(Z) &= \operatorname{SoD}(Y)/\sigma \\
d_Z([y_1]_\rho, [y_2]_\rho) &= [d_Y(y_1, y_2)]_\sigma \ .
\end{aligned}$$

The metric d_Z is well-defined because $d_Y(y_1, y_2) \sigma d_Y(y_1', y_2')$ whenever $y_1 \rho y_1'$ and $y_2 \rho y_2'$.

Let $g : Y \to Z$ be such, that $\mathrm{f}[g]$ is the natural projection from $\operatorname{Carr}(Y)$ to $\operatorname{Carr}(Y)/\rho$, and $\mathrm{c}[g]$ is the natural projection from $\operatorname{SoD}(Y)$ to $\operatorname{SoD}(Y)/\sigma$. Then clearly $g \circ f_1 = g \circ f_2$ due to the definition of ρ and σ.

Suppose that there is an arrow $g' Y \to Z'$, such that $g' \circ f_1 = g' \circ f_2$. We can then factor it through g. Define $h : Z \to Z'$ as follows:

$$\begin{aligned}
\mathrm{f}[h]([y]_\rho) &= \mathrm{f}[g'](y) \\
\mathrm{c}[h]([v]_\sigma) &= \mathrm{c}[g'](v) \ .
\end{aligned}$$

Then both $\mathrm{f}[h]$ and $\mathrm{c}[h]$ are well-defined. Indeed, if there exist $y \rho y'$, such that $\mathrm{f}[g'](y) \neq \mathrm{f}[g'](y')$, then by definition of ρ there must exist $y_0, \ldots, y_n$, such that $y = y_0$, $y_n = y'$ and for each $i \in \{1, \ldots, n\}$ there exists some $x_i \in \operatorname{Carr}(X)$, such that $\{y_{i-1}, y_i\} = \{f_1(x_i), f_2(x_i)\}$. For some i, $\mathrm{f}[g](y_{i-1}) \neq \mathrm{f}[g](y_i)$ and hence also $\mathrm{f}[g \circ f_1](x_i) \neq \mathrm{f}[g \circ f_2](x_i)$.

Consider the kernel of $\mathrm{c}[g']$; it is a congruence on the set of distances $\operatorname{SoD}(Y)$. Clearly, it must contain all pairs named in (15). Hence the kernel is a superset of σ and there are no $v \sigma v'$, such that $\mathrm{c}[g'](v) \neq \mathrm{c}[g'](v')$.

For any other $h' : Z \to Z'$, we have $h' \circ g \neq h \circ g$, because $\mathrm{f}[g]$ and $\mathrm{c}[g]$ are both onto. Hence h is unique and Z, g are a coequalizer.

Finally note that in category $\mathbf{GM}_0^\infty$, the given construction gives us arrows that preserve the distances 0 and ∞. Hence we get coequalizers also in $\mathbf{GM}_0^\infty$. □

3.3.6.4 Earth Mover's Distance for Probability Distributions. To analyze workflows, we have to understand two things: what happens with distances and sensitivities when we form products, and what happens with then when we form probability distributions. Products have been discussed previously, let us now discuss probabilities.

Let A be an (ordinary) metric space with d_A being the metric on it. Intuitively, the distance d_A is generalized to distributions $\mathcal{D}(A)$ by first thinking about a probability distribution as "dirt" piled over A, with the probability of any particular $a \in A$ showing how much dirt has been put on top of it. The distance between two probability distributions indicates the minimum effort it takes to turn the first pile into the second by moving the dirt around on top of elements of A. Formally:

Proposition 3.40 (Earth mover's distance)**.** *Let $d_A : A \times A \to \mathbb{R}_+$ be a metric on the set A. Define $\widehat{d_A} : \mathcal{D}(A) \times \mathcal{D}(A) \to \mathbb{R}_+$ as follows:*

$$\widehat{d_A}(\chi_1, \chi_2) = \inf\{ \sum_{a_1, a_2 \in A} \psi(a_1, a_2) \cdot d_A(a_1, a_2) \,|\, \psi \in \chi_1 \boxtimes \chi_2 \} \ . \tag{16}$$

Then $\widehat{d_A}$ is a metric on $\mathcal{D}(A)$.

Here ψ can be seen as the "plan for moving the dirt from" in order to make one pile similar to the other one.

Proof. We have to show that $\widehat{d_A}$ satisfies the axioms of the metric. First, if $\chi_1 = \chi_2$ then $\widehat{d_A}(\chi_1, \chi_2) = 0$, because then the distribution

$$\psi(a_1, a_2) = \begin{cases} \chi(a_1), & \text{if} a_1 = a_2 \\ 0, & \text{if} a_1 \neq a_2 \end{cases}$$

belongs to $\chi_1 \boxtimes \chi_2$ and all summands in the sum in (16) are 0. *Vice versa*, if $\chi_1 \neq \chi_2$ then any $\psi \in \chi_1 \boxtimes \chi_2$ must put non-zero weight on some pair (a_1, a_2), where $a_1 \neq a_2$.

Second, $\widehat{d_A}$ is obviously symmetric, because d_A is symmetric.

Third, $\widehat{d_A}$ satisfies the triangle inequality. For any $\psi \in \mathcal{D}(A \times A)$, let $D(\psi) = \sum_{a_1, a_2} \psi(a_1, a_2) \cdot d_A(a_1, a_2)$, where the summation is over all pairs of elements of A. We have defined $\widehat{d_A}(\chi_1, \chi_2)$ as the infimum of $D(\psi)$ over all $\psi \in \chi_1 \boxtimes \chi_2$. Let now $\chi_1, \chi_2, \chi_3 \in \mathcal{D}(A)$. Let $\psi_1 \in \chi_1 \boxtimes \chi_2$ and $\psi_2 \in \chi_2 \boxtimes \chi_3$. Define $\psi_3 \in \mathcal{D}(A \times A)$ by $\psi_3(a_1, a_3) = \sum_{a_2 \in A} \psi_1(a_1, a_2) \cdot \psi_2(a_2, a_3)/\chi_2(a_2)$. Then ψ_3 is a probability distribution. Indeed, $\psi_3(a_1, a_3) \geq 0$ for all $a_1, a_3 \in A$ and

$$\sum_{a_1, a_3} \psi_3(a_1, a_3) = \sum_{a_1, a_2, a_3} \frac{\psi_1(a_1, a_2) \cdot \psi_2(a_2, a_3)}{\chi_2(a_2)} = \sum_{a_1, a_2} \frac{\psi_1(a_1, a_2)}{\chi_2(a_2)} \cdot \sum_{a_3} \psi_2(a_2, a_3) =$$
$$\sum_{a_1, a_2} \frac{\psi_1(a_1, a_2)}{\chi_2(a_2)} \cdot \chi_2(a_2) = \sum_{a_1, a_2} \psi_1(a_1, a_2) = 1 \ .$$

Moreover we have

$$\sum_{a_1} \psi_3(a_1, a_3) = \sum_{a_1, a_2} \frac{\psi_1(a_1, a_2) \cdot \psi_2(a_2, a_3)}{\chi_2(a_2)} = \sum_{a_2} \frac{\psi_2(a_2, a_3)}{\chi_2(a_2)} \cdot \sum_{a_1} \psi_1(a_1, a_2) = \sum_{a_2} \psi_2(a_2, a_3) = \chi_3(a_3)$$

and similarly $\sum_{a_3} \psi_3(a_1, a_3) = \chi_1(a_3)$. Hence $\psi_3 \in \chi_1 \boxtimes \chi_3$ and $\widehat{d_A}(\chi_1, \chi_3) \leq D(\psi_3)$. The latter can be bounded as

$$D(\psi_3) = \sum_{a_1, a_2, a_3} \frac{\psi_1(a_1, a_2) \cdot \psi_2(a_2, a_3)}{\chi_2(a_2)} \cdot d_A(a_1, a_3) \leq$$
$$\sum_{a_1, a_2, a_3} \frac{\psi_1(a_1, a_2) \cdot \psi_2(a_2, a_3)}{\chi_2(a_2)} \cdot d_A(a_1, a_2) + \sum_{a_1, a_2, a_3} \frac{\psi_1(a_1, a_2) \cdot \psi_2(a_2, a_3)}{\chi_2(a_2)} \cdot d_A(a_2, a_3)$$

Here the first sum is equal to

$$\sum_{a_1,a_2} \frac{\psi_1(a_1,a_2)}{\chi_2(a_2)} \cdot d_A(a_1,a_2) \cdot \sum_{a_3} \psi_2(a_2,a_3) = \sum_{a_1,a_2} \psi_1(a_1,a_2) \cdot d_A(a_1,a_2) = D(\psi_1)$$

and similarly the second sum is equal to $D(\psi_2)$. We have $D(\psi_3) \leq D(\psi_1) + D(\psi_2)$ and by taking the infima of all three $D(\cdots)$-s, we obtain the triangle inequality for $\widehat{d_A}$. □

It is possible to generalize earth mover's distance to generalized metric spaces, but only if the sets of distances have some extra structure. We need to be able to multiply the distances with non-negative real numbers; this multiplication has to satisfy certain axioms, similar to axioms of vector spaces.

Definition 3.34 (Distances with multiples)**.** Let V be a set of distances. We say that V *has multiples*, if there exists an operation $\cdot : \mathbb{R}_+ \times V \to V$, satisfying the following equalities for all $a, b \in \mathbb{R}_+$ and $u, v \in V$:

$$\begin{aligned} a \cdot (b \cdot v) &= (ab) \cdot v & a \cdot (u+v) &= a \cdot u + a \cdot v \\ 0 \cdot v &= 0 & (a+b) \cdot v &= a \cdot v + b \cdot v \\ 1 \cdot v &= v & u \leq v &\Rightarrow a \cdot u \leq a \cdot v \ . \end{aligned}$$

Note that the axioms also imply $a \cdot v \leq b \cdot v$ if $a \leq b$. Indeed, we have $a \cdot v = a \cdot v + 0 \leq a \cdot v + (b-a) \cdot v = b \cdot v$.

For a partially ordered set V, let $\mathcal{F}(V)$ denote the set of all *upwards closed* subsets of V. I.e. $U \subseteq V$ is an element of $\mathcal{F}(V)$ if $u \in U$ and $u \leq v$ imply $v \in U$ for all $u, v \in V$. For an arbitrary $U \subseteq V$ we let $\uparrow U$ denote the *upwards closure* of U, i.e. the smallest upwards closed set that contains U as a subset.

If V is a set of distances, then we can also define the structure of a set of distances on $\mathcal{F}(V)$. Namely, for $U_1, U_2 \in \mathcal{F}(V)$, we define

- $U_1 + U_2 = \uparrow\{v_1 + v_2 \,|\, v_1 \in U_1, v_2 \in U_2\}$;
- $U_1 \leq U_2$ iff $U_2 \subseteq U_1$;
- $0_{\mathcal{F}(V)} = \uparrow\{0\} = V$.

It is straightforward to verify that the axioms of a set of distances (Def. 3.28) hold. Also, if V has multiplies, then $\mathcal{F}(V)$ has multiplies, too, defined simply by $a \cdot U = \uparrow\{c \cdot v \,|\, v \in U\}$.

Proposition 3.41 (Generalized earth mover's distance)**.** *Let X be a generalized metric space, such that* $\mathrm{SoD}(X)$ *has multiples. Then Y, defined as follows, is also a generalized metric space.*

- $\mathrm{Carr}(Y) = \mathcal{D}(\mathrm{Carr}(X))$*;*
- $\mathrm{SoD}(Y) = \mathcal{F}(\mathrm{SoD}(X))$*;*
- $d_Y(\chi_1, \chi_2) = \uparrow\{\sum_{x,x' \in \mathrm{Carr}(X)} \psi(x,x') \cdot d_X(x,x') \,|\, \psi \in \chi_1 \boxtimes \chi_2\}$ *for all* $\chi_1, \chi_2 \in \mathcal{D}(\mathrm{Carr}(X))$.

Proof. We have to verify that the axioms of generalized metric spaces hold. This verification is similar to the proof of Prop. 3.40. □

The notion that we have defined is indeed a generalization of earth mover's distance. We get it back if we require $\mathrm{SoD}(X) = \mathbb{R}_+$. Note that $\mathcal{F}(\mathbb{R}_+) \cong \mathbb{R}_+$.

In the rest of this section (Sec. 3.3.6.4) we are going to study how the Kleisli composition $\circ_{\mathrm{Kl}}$ interacts with sensitivities. For this purpose, we introduce the following notation:

- X, Y, Z are three sets;
- V_X, V_Y, V_Z are three sets of distances with multiples;
- d_X, d_Y, d_Z are generalized metrics on X, Y, Z, respectively, ranging over V_X, V_Y, V_Z;
- $d_{\mathcal{D}(Y)} : \mathcal{D}(Y) \times \mathcal{D}(Y) \to \mathcal{F}(V_Y)$ and $d_{\mathcal{D}(Z)} : \mathcal{D}(Z) \times \mathcal{D}(Z) \to \mathcal{F}(V_Z)$ are the generalized earth mover's distances built from d_Y and d_Z.

First we consider how the lifting of a mapping with $\mathcal{D}(\cdot)$ affects its sensitivity. If $c : V_Y \overset{\mathrm{m}}{\to} V_Z$ then we say that c is *concave* if $\sum_{i=1}^{\infty} \lambda_i c(v_i) \leq c(\sum_{i=1}^{\infty} \lambda_i v_i)$ for all $\lambda_1, \lambda_2, \ldots \in \mathbb{R}_+$ and $v_1, v_2, \ldots \in V_Y$ satisfying $\sum_{i=1}^{\infty} \lambda_i = 1$. Note that concavity generalizes superadditivity.

Proposition 3.42. *Let $f : Y \to Z$ be a mapping between two metric spaces and $c : V_Y \overset{\mathrm{m}}{\to} V_Z$ its sensitivity, such that c is concave. Then the sensitivity of the mapping $\mathcal{D}(f) : \mathcal{D}(Y) \to \mathcal{D}(Z)$ is $\mathcal{F}(c) : \mathcal{F}(V_Y) \to \mathcal{F}(V_Z)$.*

Before giving the proof, let us recall the meaning of $\mathcal{D}(f)$ and $\mathcal{F}(c)$. These mappings are given by

$$\mathcal{D}(f)(\chi) = \{z \mapsto \sum_{y \in f^{-1}(z)} \chi)(y) \,|\, z \in Z\}$$
$$\mathcal{F}(c)(U) = \uparrow\{c(v) \,|\, v \in U\} \ .$$

Proof. Let $\chi, \chi' \in \mathcal{D}(Y)$. Then

$$d_{\mathcal{D}(Z)}(\mathcal{D}(f)(\chi), \mathcal{D}(f)(\chi')) = \uparrow\{\sum_{z,z' \in Z} \psi(z,z') \cdot d_Z(z,z') \,|\, \psi \in \mathcal{D}(f)(\chi) \boxtimes \mathcal{D}(f)(\chi')\} =$$
$$\uparrow\{\sum_{y,y' \in Y} \bar{\psi}(y,y') \cdot d_Z(f(y), f(y')) \,|\, \bar{\psi} \in \chi \boxtimes \chi'\} \leq \uparrow\{\sum_{y,y' \in Y} \bar{\psi}(y,y') \cdot c(d_Y(y,y')) \,|\, \bar{\psi} \in \chi \boxtimes \chi'\} \leq$$
$$\uparrow\{c\Big(\sum_{y,y' \in Y} \bar{\psi}(y,y') \cdot d_Y(y,y')\Big) \,|\, \bar{\psi} \in \chi \boxtimes \chi'\} = \uparrow\{c(v) \,|\, v \in d_{\mathcal{D}(Y)}(\chi,\chi')\} = \mathcal{F}(c)(d_{\mathcal{D}(Y)}(\chi,\chi')) \ . \quad \square$$

We continue with studying the sensitivity of $g \circ_{\mathrm{Kl}} f$, where $f : X \to \mathcal{D}(Y)$ and $g : Y \to \mathcal{D}(Z)$. By definition, $g \circ_{\mathrm{Kl}} f = \mu \circ \mathcal{D}(g) \circ f$, where $\mu : \mathcal{D}(\mathcal{D}(Z)) \to \mathcal{D}(Z)$ is the *monadic multiplication* defined by

$$\mu(\Phi) = \{z \mapsto \sum_{\chi \in \mathcal{D}(Z)} \Phi(\chi) \cdot \chi(z) \,|\, z \in Z\} \ .$$

Note that $\mathcal{F}(\cdot)$ is also a monad and its monadic multiplication is $\bigcup : \mathcal{F}(\mathcal{F}(V)) \to \mathcal{F}(V)$.

Proposition 3.43. *The sensitivity of $\mu : \mathcal{D}(\mathcal{D}(Z)) \to \mathcal{D}(Z)$ is $\bigcup : \mathcal{F}(\mathcal{F}(V_Z)) \to \mathcal{F}(V_Z)$.*

Proof. Let $\Phi, \Phi' \in \mathcal{D}(\mathcal{D}(Z))$. We have

$$\bigcup d_{\mathcal{D}(\mathcal{D}(Z))}(\Phi, \Phi') = \bigcup \uparrow\{\sum_{\chi,\chi' \in \mathcal{D}(Z)} \Psi(\chi,\chi') \cdot d_{\mathcal{D}(Z)}(\chi,\chi') \,|\, \Psi \in \Phi \boxtimes \Phi'\} =$$
$$\bigcup \uparrow\{\sum_{\chi,\chi' \in \mathcal{D}(Z)} \uparrow\{\sum_{z,z' \in Z} \Psi(\chi,\chi') \cdot \psi(z,z') \cdot d_Z(z,z') \,|\, \psi \in \chi \boxtimes \chi'\} \,|\, \Psi \in \Phi \boxtimes \Phi'\} \quad (17)$$

and

$$d_{\mathcal{D}(Z)}(\mu(\Phi), \mu(\Phi')) = \uparrow\{\sum_{z,z' \in Z} \psi(z,z') \cdot d_Z(z,z') \,|\, \psi \in \mu(\Phi) \boxtimes \mu(\Phi')\} \ . \quad (18)$$

Consider what it means for some $v \in V_Z$ to be an element of (17). There must exist some $\Psi \in \Phi \boxtimes \Phi'$, such that

$$v \in \sum_{\chi,\chi' \in \mathcal{D}(Z)} \uparrow\{\sum_{z,z' \in Z} \Psi(\chi,\chi') \cdot \psi(z,z') \cdot d_Z(z,z') \,|\, \psi \in \chi \boxtimes \chi'\} \ . \quad (19)$$

This happens if for each $\chi, \chi' \in \mathcal{D}(Z)$, we can pick an element from the set in (19), such that they sum up into something less than or equal to v. Picking an element from this set is equivalent to picking a suitable ψ. Hence we have that v is an element of (17), if there exists $\mathbf{F} : \mathcal{D}(Z)^2 \to \mathcal{D}(Z \times Z)$, such that $\mathbf{F}(\chi,\chi')$ is an element of $\chi \boxtimes \chi'$ for all $\chi, \chi' \in \mathcal{D}(Z)$ and

$$v \geq \sum_{\chi,\chi' \in \mathcal{D}(Z)} \sum_{z,z' \in Z} \Psi(\chi,\chi') \cdot \mathbf{F}(\chi,\chi')(z,z') \cdot d_Z(z,z') \quad (20)$$

Given such Ψ and $\mathbf{F}$, define $\psi \in \mathcal{D}(Z \times Z)$ as follows:

$$\psi(z,z') = \sum_{\chi,\chi' \in \mathcal{D}(Z)} \Psi(\chi,\chi') \cdot \mathbf{F}(\chi,\chi')(z,z') \ .$$

This is indeed a probability distribution, as we can easily verify:

$$\sum_{z,z' \in Z} \psi(z,z') = \sum_{\chi,\chi' \in \mathcal{D}(Z)} \Psi(\chi,\chi') \cdot \sum_{z,z' \in Z} \mathbf{F}(\chi,\chi')(z,z') = \sum_{\chi,\chi' \in \mathcal{D}(Z)} \Psi(\chi,\chi') \cdot 1 = 1 \ .$$

Consider now the first projection of ψ. Let $z \in Z$.

$$\sum_{z' \in Z} \psi(z,z') = \sum_{\chi,\chi' \in \mathcal{D}(Z)} \Psi(\chi,\chi') \cdot \sum_{z' \in Z} \mathbf{F}(\chi,\chi')(z,z') = \sum_{\chi,\chi' \in \mathcal{D}(Z)} \Psi(\chi,\chi') \cdot \chi(z) = \sum_{\chi \in \mathcal{D}(Z)} \Phi(\chi) \cdot \chi(z) = \mu(\Phi)(z)$$

and symmetrically, the second projection of ψ is equal to $\mu(\Phi')$. Hence $\psi \in \mu(\Phi) \boxtimes \mu(\Phi')$. Comparing (18) and (20), we conclude that v is an element of (18).

Thus $\bigcup d_{\mathcal{D}(\mathcal{D}(Z))}(\Phi,\Phi') \subseteq d_{\mathcal{D}(Z)}(\mu(\Phi),\mu(\Phi'))$, i.e. $d_{\mathcal{D}(Z)}(\mu(\Phi),\mu(\Phi')) \leq \bigcup d_{\mathcal{D}(\mathcal{D}(Z))}(\Phi,\Phi')$ according to the ordering on $\mathcal{F}(V_Z)$. □

As a simple corollary of Prop. 3.32, Prop. 3.42 and Prop. 3.43 we now get:

Proposition 3.44. *Let $f : X \to \mathcal{D}(Y)$ and $g : Y \to \mathcal{D}(Z)$ have sensitivities $c : V_X \xrightarrow{\mathrm{m}} \mathcal{F}(V_Y)$ and $c' : V_Y \xrightarrow{\mathrm{m}} \mathcal{F}(V_Z)$, respectively, where c' is concave. Then the sensitivity of $g \circ_{\mathrm{Kl}} f$ is $c' \circ_{\mathrm{Kl}} c$.*

Proof. According to the propositions proved before, the sensitivity of $g \circ_{\mathrm{Kl}} f$ is $\bigcup \circ \mathcal{F}(c') \circ c$, which is the Kleisli composition of c and c'. □

We are left to investigate how the concavity and Kleisli composition interact.

Proposition 3.45. *If $c : V_X \xrightarrow{\mathrm{m}} \mathcal{F}(V_Y)$ and $c' : V_Y \xrightarrow{\mathrm{m}} \mathcal{F}(V_Z)$ are concave, then $c' \circ_{\mathrm{Kl}} c : V_X \xrightarrow{\mathrm{m}} \mathcal{F}(V_Z)$ is also concave.*

Proof. Indeed, let $v_1, v_2, \ldots \in V_X$, $\lambda_1, \lambda_2, \ldots \in \mathbb{R}_+$, $\sum_{i=1}^{\infty} \lambda_i = 1$. Denote $\mathbb{C} = c(\sum_{i=1}^{\infty} \lambda_i v_i)$ and $\mathbb{C}' = \sum_{i=1}^{\infty} \lambda_i c(v_i)$. By concavity of c, we have $\mathbb{C} \geq \mathbb{C}'$ or $\mathbb{C} \subseteq \mathbb{C}'$. We get (recall that "$\subseteq$" is "$\geq$")

$$(c' \circ_{\mathrm{Kl}} c)\Big(\sum_{i=1}^{\infty} \lambda_i v_i\Big) = \bigcup_{u \in \mathbb{C}} c'(u) \subseteq \bigcup_{u \in \mathbb{C}'} c'(u) = \bigcup_{u_1 \in c(v_1)} \bigcup_{u_2 \in c(v_2)} \cdots c'\Big(\sum_{i=1}^{\infty} \lambda_i u_i\Big) \subseteq$$

$$\bigcup_{u_1 \in c(v_1)} \bigcup_{u_2 \in c(v_2)} \cdots \sum_{i=1}^{\infty} \lambda_i c'(u_i) = \sum_{i=1}^{\infty} \lambda_i \bigcup_{u_i \in c(v_i)} c'(u_i) = \sum_{i=1}^{\infty} \lambda_i (c' \circ_{\mathrm{Kl}} c)(v_i) \ . \quad \square$$

3.3.6.5 Set-of-Sets Distance for Probability Distributions. The simple statements of the results in Sec. 3.3.6.4 show that earth mover's distance is a natural extension of generalized metrics and sensitivities to probability distributions. Unfortunately, the metrics (over probability distributions) relevant for differential privacy are not earth mover's distances. Intuitively, the earth mover's distance characterizes the "average" distance between two probability distributions, while differential privacy is concerned with the "worst-case" distance.

For a partially ordered set V, let $\mathcal{I}(V)$ denote the set of all its *downwards closed* non-empty subsets. For a probability distribution $\chi \in \mathcal{D}(X)$, let $\operatorname{supp}\chi \subseteq X$ denote its *support*, i.e. $\operatorname{supp}\chi = \{x \in X \,|\, \chi(x) > 0\}$. If $f : X \to \mathcal{D}(Y)$ is a mapping, then let $\overline{f} : \mathcal{D}(X) \to \mathcal{D}(Y)$ denote its lifting to $\mathcal{D}(X)$; formally, $\overline{f} = \mu \circ \mathcal{D}(f)$. The following proposition gives a suitable lifting of an arbitrary generalized metric to probability distributions and its relation to differential privacy.

Proposition 3.46 (Set-of-sets distance)**.** *Let (X, d_X, V_X) be a generalized metric space. Then $(\mathcal{D}(X), d_{\mathcal{D}(X)}, \mathcal{F}(\mathcal{I}(V_X)))$ is also a generalized metric space, where*

$$d_{\mathcal{D}(X)}(\chi_1, \chi_2) = \uparrow\{\downarrow\{d_X(x, x') \,|\, (x, x') \in \operatorname{supp}\psi\} \,|\, \psi \in \chi_1 \boxtimes \chi_2\}$$

for all $\chi_1, \chi_2 \in \mathcal{D}(X)$. Moreover, if f is a mapping from a generalized metric space (X, d_X, V_X) to an (ordinary) metric space $(\mathcal{D}(Y), d_{\mathrm{dp}}, \mathbb{R}_+)$ with the sensitivity $\varepsilon : V_X \overset{\mathrm{m}}{\to} \mathbb{R}_+$, then the sensitivity of $\overline{f}$ is

$$E(\mathcal{V}) = \min_{U \in \mathcal{V}} \max_{v \in U} \varepsilon(v)$$

for all $\mathcal{V} \in \mathcal{F}(\mathcal{I}(V_X))$.

We see that compared to Prop. 3.41, we are now taking the set of all distances, instead of finding their linear combination. Downwards closure corresponds to the ordering of such subsets for the purposes of computing the sensitivity — we are only interested in the maximal elements of each such set. Similarly to $\mathcal{F}(V)$, the set $\mathcal{I}(V)$ can also be given the structure of a set of distances, if V has it: addition is again defined pointwise and $0_{\mathcal{I}(V)} = \{0\}$. A bit differently, the ordering $\leq$ for $\mathcal{I}(V)$ coincides with the subset relation $\subseteq$.

Proof. It is straightforward to verify that $d_{\mathcal{D}(X)}$ is a generalized metric; the proof follows along the lines of the proof of Prop. 3.40. For investigating the sensitivity of $\overline{f}$, let $\chi, \chi' \in \mathcal{D}(X)$, $\psi \in \chi \boxtimes \chi'$ and $y \in Y$. Then

$$\begin{aligned}
\Pr[\overline{f}(\chi) = y] = \sum_{x \in X} \chi(x) \cdot \Pr[f(x) = y] = \sum_{x, x' \in X} \psi(x, x') \cdot \Pr[f(x) = y] \leq \\
\sum_{x, x' \in X} \psi(x, x') \cdot e^{\varepsilon(d_X(x, x'))} \Pr[f(x') = y] \leq \sum_{x, x' \in X} \psi(x, x') \cdot e^{\max_{x \in \operatorname{supp}\psi(\cdot, x')} \varepsilon(d_X(x, x'))} \Pr[f(x') = y] = \\
\sum_{x' \in X} \chi'(x') \cdot e^{\max_{x \in \operatorname{supp}\psi(\cdot, x')} \varepsilon(d_X(x, x'))} \Pr[f(x') = y] \leq e^{\max_{x, x' \in \operatorname{supp}\psi} \varepsilon(d_X(x, x'))} \cdot \sum_{x' \in X} \chi'(x') \cdot \Pr[f(x') = y] = \\
e^{\max_{x, x' \in \operatorname{supp}\psi} \varepsilon(d_X(x, x'))} \cdot \Pr[\overline{f}(\chi') = y],
\end{aligned}$$

where $\operatorname{supp}\psi(\cdot, x')$ denotes the set of all $x \in X$, such that $\psi(x, x') > 0$. We obtain

$$d_{\mathrm{dp}}(\overline{f}(\chi), \overline{f}(\chi')) = \max_{y \in Y} \left| \ln \frac{\Pr[\overline{f}(\chi') = y]}{\Pr[\overline{f}(\chi) = y]} \right| \leq \min_{\psi \in \chi \boxtimes \chi'} \max_{x, x' \in \operatorname{supp}\psi} \varepsilon(d_X(x, x')) = \min_{U \in d_{\mathcal{D}(X)}(\chi, \chi')} \max_{v \in U} \varepsilon(v) \ . \quad \square$$

We can again show that the operations with probability distribution monad are matched in the set-of-sets monad. There are analogues to Prop. 3.42, Prop. 3.43 and Prop. 3.44. Let X, Y, Z, V_X, V_Y, V_Z, d_X, d_Y, d_Z be as in Sec. 3.3.6.4. Let $d_{\mathcal{D}(Y)} : \mathcal{D}(Y) \times \mathcal{D}(Y) \to \mathcal{F}(\mathcal{I}(V_Y))$ and $d_{\mathcal{D}(Z)} : \mathcal{D}(Z) \times \mathcal{D}(Z) \to \mathcal{F}(\mathcal{I}(V_Z))$ be set-of-sets distances on $\mathcal{D}(Y)$ and $\mathcal{D}(Z)$.

Proposition 3.47. *Let $f : Y \to Z$ be a mapping between two metric spaces and $c : V_Y \overset{\mathrm{m}}{\to} V_Z$ its sensitivity. Then the sensitivity of the mapping $\mathcal{D}(f) : \mathcal{D}(Y) \to \mathcal{D}(Z)$ is $\mathcal{F}(\mathcal{I}(c))$.*

Let $U \in \mathcal{F}(\mathcal{I}(V_Y))$. By definition,

$$\mathcal{F}(\mathcal{I}(c))(U) = \uparrow\{\downarrow\{c(v) \,|\, v \in \mathbf{v}\} \,|\, \mathbf{v} \in U\} \ .$$

Proof. Let $\chi, \chi' \in \mathcal{D}(Y)$. Then

$$\begin{aligned}
d_{\mathcal{D}(Z)}(\mathcal{D}(f)(\chi), \mathcal{D}(f)(\chi')) = \uparrow\{\downarrow\{d_Z(z, z') \,|\, (z, z') \in \operatorname{supp}\psi\} \,|\, \psi \in \mathcal{D}(f)(\chi) \boxtimes \mathcal{D}(f)(\chi')\} = \\
\uparrow\{\downarrow\{d_Z(f(y), f(y')) \,|\, (y, y') \in \operatorname{supp}\psi^\circ\} \,|\, \psi^\circ \in \chi \boxtimes \chi'\} \leq \uparrow\{\downarrow\{c(d_Y(y, y')) \,|\, (y, y') \in \operatorname{supp}\psi^\circ\} \,|\, \psi^\circ \in \chi \boxtimes \chi'\} = \\
\mathcal{F}(\mathcal{I}(c))(\uparrow\{\downarrow\{d_Y(y, y') \,|\, (y, y') \in \operatorname{supp}\psi^\circ\} \,|\, \psi^\circ \in \chi \boxtimes \chi'\}) = \mathcal{F}(\mathcal{I}(c))(d_{\mathcal{D}(Y)}(\chi, \chi')) \ . \quad \square
\end{aligned}$$

Before studying the sensitivity of $\mu : \mathcal{D}(\mathcal{D}(Z)) \to \mathcal{D}(Z)$, let us discuss, what is the monadic multiplication for $\mathcal{F}(\mathcal{I}(\cdot))$. It must map from $\mathcal{F}(\mathcal{I}(\mathcal{F}(\mathcal{I}(V_Z))))$ to $\mathcal{F}(\mathcal{I}(V_Z))$. For an arbitrary partially ordered set V, consider the mapping $\delta_V : \mathcal{I}(\mathcal{F}(V)) \to \mathcal{F}(\mathcal{I}(V))$, defined as follows:

$$\delta_V(\{U_1, U_2, \ldots\}) = \uparrow\{\downarrow\{v_1, v_2, \ldots\} \,|\, v_1 \in U_1, v_2 \in U_2, \ldots\}$$

Here $U_1, U_2, \ldots \in \mathcal{F}(V)$. As the argument of δ_V is downwards closed, we have that if some U_i is an element of this argument, then all supersets of U_i are also elements. We can think of the argument of δ_V as a formal "maximum of minima". The result of δ_V is a formal "minimum of maxima" that has the same value. I.e. the elements of V are considered as uninterpreted values. We take the minimum of each U_i; it is natural to consider the sets U_i upwards closed, because the minimum is not affected by extra larger elements. We then take the maximum of all minima to obtain the argument to δ_V. In the result of δ_V, we first take the maxima; these are taken over sets consisting of a single element from each U_i. Finally, we take the minimum of the maxima of all such sets.

The monadic multiplication $\mathbf{M} : \mathcal{F}(\mathcal{I}(\mathcal{F}(\mathcal{I}(V_Z)))) \to \mathcal{F}(\mathcal{I}(V_Z))$ is

$$\mathbf{M} = \mathcal{F}(\bigcup) \circ \bigcup \circ \mathcal{F}(\delta_{\mathcal{I}(V_Z)}) \ . \tag{21}$$

Let us explain that construction. We have $\mathcal{F}(\delta_{\mathcal{I}(V_Z)}) : \mathcal{F}(\mathcal{I}(\mathcal{F}(\mathcal{I}(V_Z)))) \to \mathcal{F}(\mathcal{F}(\mathcal{I}(\mathcal{I}(V_Z))))$, i.e. the first step in the construction of $\mathbf{M}$ "swaps the two middle layers". The following $\bigcup$ collapses the two $\mathcal{F}(\cdot)$-layers into one, and the final $\mathcal{F}(\bigcup)$ does the same with the two $\mathcal{I}(\cdot)$-layers.

Proposition 3.48. *The sensitivity of $\mu : \mathcal{D}(\mathcal{D}(Z)) \to \mathcal{D}(Z)$ is $\mathbf{M} : \mathcal{F}(\mathcal{I}(\mathcal{F}(\mathcal{I}(V_Z)))) \to \mathcal{F}(\mathcal{I}(V_Z))$.*

Proof. Let $\mathbf{F}$ be the following set of functions $f : \mathcal{D}(Z) \times \mathcal{D}(Z) \to \mathcal{D}(Z \times Z)$:

$$\mathbf{F} = \{f : \mathcal{D}(Z) \times \mathcal{D}(Z) \to \mathcal{D}(Z \times Z) \,|\, \forall(\chi, \chi') : f(\chi, \chi') \in \chi \boxtimes \chi'\} \ .$$

Let $\Phi, \Phi' \in \mathcal{D}(\mathcal{D}(Z))$. We have

$$\begin{aligned}
&\mathbf{M}(d_{\mathcal{D}(\mathcal{D}(Z))}(\Phi, \Phi')) = \mathbf{M}(\uparrow\{\downarrow\{d_{\mathcal{D}(Z)}(\chi, \chi') \,|\, (\chi, \chi') \in \operatorname{supp} \Psi\} \,|\, \Psi \in \Phi \boxtimes \Phi'\}) = \\
&\qquad \mathbf{M}(\uparrow\{\downarrow\{\uparrow\{\downarrow\{d_Z(z, z') \,|\, (z, z') \in \operatorname{supp} \psi\} \,|\, \psi \in \chi \boxtimes \chi'\} \,|\, (\chi, \chi') \in \operatorname{supp} \Psi\} \,|\, \Psi \in \Phi \boxtimes \Phi'\}) = \\
&(\mathcal{F}(\bigcup) \circ \bigcup)(\uparrow\{\uparrow\{\downarrow\{\downarrow\{d_Z(z, z') \,|\, (z, z') \in \operatorname{supp} f(\chi, \chi')\} \,|\, (\chi, \chi') \in \operatorname{supp} \Psi\} \,|\, f \in \mathbf{F}\} \,|\, \Psi \in \Phi \boxtimes \Phi'\}) = \\
&\qquad \uparrow\{\downarrow\{d_Z(z, z') \,|\, (\chi, \chi') \in \operatorname{supp} \Psi, (z, z') \in \operatorname{supp} f(\chi, \chi')\} \,|\, f \in \mathbf{F}, \Psi \in \Phi \boxtimes \Phi'\}
\end{aligned} \tag{22}$$

and

$$d_{\mathcal{D}(Z)}(\mu(\Phi), \mu(\Phi')) = \uparrow\{\downarrow\{d_Z(z, z') \,|\, (z, z') \in \operatorname{supp} \psi\} \,|\, \psi \in \mu(\Phi) \boxtimes \mu(\Phi')\} \ . \tag{23}$$

Consider what it means for some $U \in \mathcal{I}(V_Z)$ to be an element of (22). There must exist $\Psi \in \Phi \boxtimes \Phi'$ and $f \in \mathbf{F}$ so, that

$$\downarrow\{d_Z(z, z') \,|\, (\chi, \chi') \in \operatorname{supp} \Psi, (z, z') \in \operatorname{supp} f(\chi, \chi')\} \subseteq U \ . \tag{24}$$

Let $\psi \in \mathcal{D}(Z \times Z)$ be the following probability distribution:

$$\psi(z, z') = \sum_{\chi, \chi' \in \mathcal{D}(Z)} \Psi(\chi, \chi') \cdot f(\chi, \chi')(z, z') \ . \tag{25}$$

This is indeed a probability distribution, one can verify it in the same way as in the proof of Prop. 3.43. We can continue in the same way is in the proof of Prop. 3.43 and find that $\psi \in \mu(\Phi) \boxtimes \mu(\Phi')$.

We have

$$\operatorname{supp} \psi \subseteq \bigcup_{(\chi, \chi') \in \operatorname{supp} \Psi} \operatorname{supp} f(\chi, \chi') \ . \tag{26}$$

Indeed, let $(z, z') \in \operatorname{supp}(\psi)$. By (25), there exist $(\chi, \chi') \in \operatorname{supp} \Psi$, such that $f(\chi, \chi')(z, z') > 0$. This is exactly what is required by the right hand side of (26).

Applying d_Z to both sides of (26) and taking the downwards closure, we obtain

$$\downarrow\{d_Z(z,z') \,|\, (z,z') \in \operatorname{supp}\psi\} \subseteq \downarrow\{d_Z(z,z') \,|\, (\chi,\chi') \in \operatorname{supp}\Psi, (z,z') \in \operatorname{supp} f(\chi,\chi')\} \ .$$

Together with (23) and (24), this implies that U is an element of (23). Thus $\mathbf{M}(d_{\mathcal{D}(\mathcal{D}(Z))}(\Phi,\Phi')) \subseteq d_{\mathcal{D}(Z)}(\mu(\Phi),\mu(\Phi'))$, i.e. $d_{\mathcal{D}(Z)}(\mu(\Phi),\mu(\Phi')) \leq \mathbf{M}(\bigcup d_{\mathcal{D}(\mathcal{D}(Z))}(\Phi,\Phi'))$ according to the ordering on $\mathcal{F}(\mathcal{I}(V_Z))$. □

An immediate corollary is

Proposition 3.49. *Let $f : X \to \mathcal{D}(Y)$ and $g : Y \to \mathcal{D}(Z)$ have sensitivities $c : V_X \xrightarrow{\mathrm{m}} \mathcal{F}(\mathcal{I}(V_Y))$ and $c' : V_Y \xrightarrow{\mathrm{m}} \mathcal{F}(\mathcal{I}(V_Z))$, respectively. Then the sensitivity of $g \circ_{\mathrm{Kl}} f$ is $c' \circ_{\mathrm{Kl}} c$.*

Proof. Same as Prop. 3.44. □

3.3.6.6 Normal Distances. The distance defined in Prop. 3.46 is useful if we have a mapping to $\mathcal{D}(Y)$, where we are interested in the distance d_{dp}. This typically happens at the last stages of a workflow. In other positions, we have a component that implements some mapping $f : X \to \mathcal{D}(Y)$ with the sensitivity c for some generalized metrics on X and $\mathcal{D}(Y)$. The sensitivity of lifted mapping $\overline{f} : \mathcal{D}(X) \to \mathcal{D}(Y)$ is somehow related to c; the exact relationship depends on the generalized metric we consider on $\mathcal{D}(X)$. To analyze a workflow, we need to compose f in parallel with an identity function (actually: the monadic unit) $\eta_Z : Z \to \mathcal{D}(Z)$, and find the sensitivity of $\overline{f \times \eta_Z} : \mathcal{D}(X \times Z) \to \mathcal{D}(Y \times Z)$. Given these sensitivities, Prop. 3.32 allows us to find the sensitivity of the entire workflow. We can find this sensitivity if the distances satisfy some mild conditions.

Definition 3.35 (Normal metric)**.** Let $d_{\mathcal{D}(X)}$ be a generalized metric on $\mathcal{D}(X)$, using the set of distances $V_{\mathcal{D}(X)}$. We say that $d_{\mathcal{D}(X)}$ is a *normal metric*, if there exists a set of distances $V^\flat_{\mathcal{D}(X)}$ and a mapping $d^\flat_{\mathcal{D}(X)} : \mathcal{D}(X \times X) \to V^\flat_{\mathcal{D}(X)}$, such that

$$\begin{aligned} V_{\mathcal{D}(X)} &= \mathcal{F}(V^\flat_{\mathcal{D}(X)}) \\ d_{\mathcal{D}(X)}(\chi,\chi') &= \uparrow\{d^\flat_{\mathcal{D}(X)}(\psi) \,|\, \psi \in \chi \boxtimes \chi'\} \\ d^\flat\mathcal{D}(X)(\psi_3) &\leq d^\flat_{\mathcal{D}(X)}(\psi_1) + d^\flat_{\mathcal{D}(X)}(\psi_2), \end{aligned}$$

where the last inequality holds for all $\psi_1, \psi_2 \in \mathcal{D}(X \times X)$ satisfying $\psi_1\downarrow_2 = \psi_2\downarrow_1 =: \phi$ and ψ_3 is defined by $\psi_3(x_1,x_3) = \sum_{x_2 \in X} \psi_1(x_1,x_2)\psi_2(x_2,x_3)/\phi(x_2)$.

If $d_{\mathcal{D}(X)}$ is a normal metric then we say that $(\mathcal{D}(X), d_{\mathcal{D}(X)}, V_{\mathcal{D}(X)})$ is a *normal metric space*.

Clearly, earth mover's distance (Prop 3.41) is a normal distance. Also, the differential privacy distance d_{dp} is normal. Indeed, for any set X, define $d^\flat_{\mathrm{dp}} : \mathcal{D}(X \times X) \to \mathbb{R}_+$ simply by $d^\flat_{\mathrm{dp}}(\psi) = d_{\mathrm{dp}}(\psi\downarrow_1, \psi\downarrow_2)$. The codomain of d_{dp} is $\mathbb{R}_+ \cong \mathcal{F}(\mathbb{R}_+)$ and $d_{\mathrm{dp}}(\chi,\chi') = \inf_{\psi \in \chi \boxtimes \chi'} d^\flat_{\mathrm{dp}}(\psi)$.

Proposition 3.50 (Product of normal metric spaces)**.** *Let $(\mathcal{D}(X), d_{\mathcal{D}(X)}, \mathcal{F}(V^\flat_{\mathcal{D}(X)}))$ and $(\mathcal{D}(Y), d_{\mathcal{D}(Y)}, \mathcal{F}(V^\flat_{\mathcal{D}(Y)}))$ be normal metric spaces. Then $(\mathcal{D}(X \times Y), d_{\mathcal{D}(X\times Y)}, \mathcal{F}(V^\flat_{\mathcal{D}(X)} \times V^\flat_{\mathcal{D}(Y)}))$ is also a normal metric space, where*

$$d^\flat_{\mathcal{D}(X\times Y)}(\psi) = (d^\flat_{\mathcal{D}(X)}(\psi\downarrow_{1,3}), d^\flat_{\mathcal{D}(Y)}(\psi\downarrow_{2,4})) \ .$$

Here $d^\flat_{\mathcal{D}(X\times Y)} : \mathcal{D}(X \times Y \times X \times Y) \to V^\flat_{\mathcal{D}(X)} \times V^\flat_{\mathcal{D}(Y)}$. I.e. the argument of $d^\flat_{\mathcal{D}(X\times Y)}$ is a probability distribution over quadruples ψ. In the definition, we project it once to its first and third component (giving an element of $\mathcal{D}(X \times X)$), and once to its second and fourth component (giving an element of $\mathcal{D}(Y \times Y)$).

Proof. It is straightforward to verify that $d_{\mathcal{D}(X\times Y)}$ satisfies the axioms of a generalized metric. □

For the purposes of the proof of the next proposition, let us define how to apply the function $f : X \to \mathcal{D}(Z)$ to a probability distribution $\phi \in \mathcal{D}(X \times X)$. There is no single way to apply it, and as a result we obtain a set of probability distributions $\mathcal{A}_f(\phi) \subseteq \mathcal{D}(Z \times Z)$. The elements of this set are obtained by selecting a probability distribution $\psi_{x,x'} \in f(x) \boxtimes f(x')$ for each pair $x, x' \in X$, and then summing them up according to ϕ. Formally,

$$\mathcal{A}_f(\phi) = \{\{(z,z') \mapsto \sum_{x,x' \in X} \phi(x,x') \cdot \psi_{x,x'}(z,z') \,|\, z,z' \in Z\} \,|\, x,x' \in X, \psi_{x,x'} \in f(x) \boxtimes f(x')\} \ .$$

Suppose $\chi, \chi' \in \mathcal{D}(X)$ and $\phi \in \chi \boxtimes \chi'$. Then $\mathcal{A}_f(\phi) \subseteq \overline{f}(\chi) \boxtimes \overline{f}(\chi')$. The opposite also holds: if $\psi \in \overline{f}(\chi) \boxtimes \overline{f}(\chi')$, then there exists some $\phi \in \chi \boxtimes \phi'$, such that $\psi \in \mathcal{A}_f(\phi)$. To see this, consider the function $g : X \to \mathcal{D}(X \times Z)$, defined for all $x, x' \in X$ and $z \in Z$ by

$$Pr[g(x) = (x', z)] = \begin{cases} Pr[f(x) = z], & \text{if} x = x' \\ 0, & \text{otherwise.} \end{cases}$$

Clearly, for each $\psi \in \overline{f}(\chi) \boxtimes \overline{f}(\chi') \subseteq \mathcal{D}(Z \times Z)$ there exists some $\psi^\circ \in \overline{g}(\chi) \boxtimes \overline{g}(\chi') \subseteq \mathcal{D}(X \times Z \times X \times Z))$, such that $\psi = \psi^\circ{\downarrow_{2,4}}$. Now take $\phi = \psi^\circ{\downarrow_{1,3}}$.

Proposition 3.51 (Parallel composition in normal metric spaces)**.** *Let (X, d_X, V_X) be a generalized metric space. Let $(\mathcal{D}(Y), d_{\mathcal{D}(Y)}, \mathcal{F}(V_Y))$ and $(\mathcal{D}(Z), d_{\mathcal{D}(Z)}, \mathcal{F}(V_Z))$ be normal metric spaces. Let $f : X \to \mathcal{D}(Z)$ have sensitivity $c : V_X \xrightarrow{\text{m}} \mathcal{F}(V_Z)$. Let $(\mathcal{D}(X), d_{\mathcal{D}(X)}, \mathcal{F}(V'_X))$ be a normal metric space. Let the sensitivity of $\overline{f} : \mathcal{D}(X) \to \mathcal{D}(Z)$ be $\overline{c}(U) = \bigcup_{v \in U} C(v)$ for $U \in \mathcal{F}(V'_X)$ and $C : V'_X \to \mathcal{F}(V_Z)$. Then the sensitivity of $\overline{f \times \eta_Y} : \mathcal{D}(X \times Y) \to \mathcal{D}(Z \times Y)$ is*

$$c'(\mathcal{V}) = \bigcup_{(v,u) \in \mathcal{V}} C(v) \times \uparrow\{u\}$$

for any $\mathcal{V} \in \mathcal{F}(V'_X \times V_Y)$.

Proof. Let $\phi, \phi' \in \mathcal{D}(X \times Y)$ and $(v,u) \in d_{\mathcal{D}(X \times Y)}(\phi, \phi')$. It is sufficient for us to show that $(v^\#, u^\#) \in d_{\mathcal{D}(Z \times Y)}(\overline{f \times \eta_Y}(\phi), \overline{f \times \eta_Y}(\phi'))$ for any $v^\# \in C(v)$ and $u^\# \geq u$. We are thus looking for a $\psi^\# \in \overline{f \times \eta_Y}(\phi) \boxtimes \overline{f \times \eta_Y}(\phi')$, such that $(v^\#, u^\#) \geq d^\flat_{\mathcal{D}(Z \times Y)}(\psi^\#) \in V_Z \times V_Y$.

Consider the set of probability distributions $\bigcup\{\mathcal{A}_f(\rho) \,|\, \rho \in \phi{\downarrow_1} \boxtimes \phi'{\downarrow_1}\}$. It contains all distributions $\rho^\# \in \mathcal{D}(Z \times Z)$ that are used to define $d_{\mathcal{D}(Z)}(\overline{f}(\phi{\downarrow_1}), \overline{f}(\phi'{\downarrow_1}))$. It hence also contains a distribution $\rho^\#$, such that $v^\# \in d^\flat_{\mathcal{D}(Z)}(\rho^\#)$.

We can similarly consider the set of probability distributions $\bigcup\{\mathcal{A}_{f \times \eta_Y}(\psi) \,|\, \psi \in \phi \boxtimes \phi'\}$. The distributions $\rho^\#$ are obtained from these distributions $\psi^\#$ by projecting onto the first and third components. At the same time, the distributions in $\phi{\downarrow_2} \boxtimes \phi'{\downarrow_2}$ are obtained from the distributions $\psi^\#$ by projecting onto the second and fourth components. There exists a distribution $\psi^\#$ in this set, such that $({\downarrow_{1,3}}\psi^\#) = \rho^\#$ and $d^\flat_{\mathcal{D}(Y)}(\psi^\#{\downarrow_{2,4}}) = u$. This is the distribution $\psi^\#$ that we were looking for. □

3.3.6.7 Distances and Sensitivities for Streams of Data. For a set X let $\mathcal{S}(X)$ denote the set of *streams* over X, i.e. the set of infinite sequences over X. If we define the lifting of a mapping $f : X \to Y$ to $\mathcal{S}(f) : \mathcal{S}(X) \to \mathcal{S}(Y)$ pointwise, then $\mathcal{S}(\cdot)$ is a functor. If (X, d_X, V_X) is a generalized metric space, then $(\mathcal{S}(X), d_{\mathcal{S}(X)}, \mathcal{S}(V_X))$ is also a generalized metric space, where $d_{\mathcal{S}(X)}$ is defined pointwise (i.e. both arguments of $d_{\mathcal{S}(X)}$ are assumed to *move at the same pace*).

A stream $\mathbf{x} \in \mathcal{S}(X)$ can be decomposed to its *head* — an element of X —, and its *tail* — a stream over X again. We write $\mathbf{x} = x : \mathbf{x}'$ to denote that x is the head of $\mathbf{x}$ and $\mathbf{x}'$ is the tail of $\mathbf{x}$.

If $f : X \to \mathcal{D}(Y)$ is ε-differentially private, then the differential privacy level of $\mathcal{S}(f) : \mathcal{S}(X) \to \mathcal{S}(\mathcal{D}(Y)) \subsetneq \mathcal{D}(\mathcal{S}(Y))$ depends on how we assume the adversary to differentiate between different streams of values from Y. However, it is natural to assume that the level of distinction between $\mathcal{S}(f)(\mathbf{x}_1)$ and

$\mathcal{S}(f)(\mathbf{x}_2)$ is defined solely in terms of the distances between the corresponding elements in the resulting streams. This justifies using $\mathcal{S}(V_X)$ as the set of distances for $\mathcal{S}(X)$.

A *(synchronous) stream processor*, transforming a stream $\mathbf{x} \in \mathcal{S}(X)$ to $\mathbf{y} \in \mathcal{S}(Y)$ consists of a set of states S, the initial state $s_0 \in S$, and a mapping $f : S \times X \to S \times Y$. The stream processor works by reading the next element of $\mathbf{x}$, applying f to it and the current state, receiving back the next element of $\mathbf{y}$ and the next state. We call it synchronous because it produces exactly one element of $\mathbf{y}$ for each element of $\mathbf{x}$ that it reads. This restriction is natural because we think of the streams we are working with to be indexed by time moments. Formally, given S, s_0 and f, we define $\widetilde{f^{[s_0]}} : \mathcal{S}(X) \to \mathcal{S}(Y)$ by

$$\widetilde{f^{[s_0]}}(x : \mathbf{x}) = \textbf{let } (y, s_1) = f(s_0, x) \textbf{ in } y : \widetilde{f^{[s_1]}}(\mathbf{x}) \ .$$

If (X, d_X, V_X) and (Y, d_Y, V_Y) are generalized metric spaces, then we can also define the structure of a generalized metric space on their Cartesian product. It results in $(X \times Y, d_{X\times Y}, V_X \times V_Y)$, where the ordering and addition on $V_X \times V_Y$ is defined pointwise, and $d_{X\times Y}$ simply applies both d_X and d_Y. If the sensitivity of the mapping f defining a stream processor is $c_f : V_S \times V_X \to V_S \times V_Y$, then the sensitivity of $\widetilde{f^{[s_0]}}$ is simply $\widetilde{c_f{}^{[0]}}$.

In practice, it will be more likely that the type of our stream processors is $f : S \times X \to \mathcal{D}(S \times Y)$, i.e. the stream processor is probabilistic and defines a function $\widetilde{f^{[s_0]}} : \mathcal{S}(X) \to \mathcal{D}(\mathcal{S}(Y))$. If this is the case, and the sensitivity of f is given by the mapping $c_f : V_S \times V_X \to \mathcal{F}(\mathcal{I}(V_S \times V_Y))$, then the sensitivity c' of $\widetilde{f^{[s_0]}}$ is the following. Define the mapping $c^{\#} : V_S \times \mathcal{S}(V_X) \to \mathcal{F}(\mathcal{I}(\mathcal{S}(V_Y)))$ through the following *co-inductive* construction. In order to compute $c^{\#}(w_0, x : \mathbf{x})$,

1. let $\mathbf{X} = c(w_0, x) \in \mathcal{F}(\mathcal{I}(V_S \times V_Y))$;
2. let $\mathfrak{X} = \mathcal{F}(\mathcal{I}(g))(\mathbf{X}) \in \mathcal{F}(\mathcal{I}(V_Y \times \mathcal{F}(\mathcal{I}(\mathcal{S}(V_Y)))))$, where $g : V_S \times V_Y \to V_Y \times \mathcal{F}(\mathcal{I}(\mathcal{S}(V_Y)))$ is defined by $g(v_s, v_y) = (v_y, c^{\#}(v_s, \mathbf{x}))$;
3. let $\mathfrak{Y} = \mathcal{F}(\mathcal{I}(h))(\mathfrak{X}) \in \mathcal{F}(\mathcal{I}(\mathcal{F}(\mathcal{I}(\mathcal{S}(V_Y)))))$, where $h : V_Y \times \mathcal{F}(\mathcal{I}(\mathcal{S}(V_Y))) \to \mathcal{F}(\mathcal{I}(\mathcal{S}(V_Y)))$ is defined by $h(v_y, \mathbf{Y}) = \mathcal{F}(\mathcal{I}(v_y : \cdot))(\mathbf{Y})$, i.e. h prepends v_y to all streams in $\mathbf{Y}$;
4. return $\mu(\mathfrak{Y})$, where μ is the monad multiplication for $\mathcal{F}(\mathcal{I}(\cdot))$.

Then $c' = c^{\#}(0, \cdot)$. The construction of $c^{\#}$ is well-founded because it is recursively applied only to components of its arguments.

If c is sufficiently simple (e.g. it is guaranteed to return only finitely generated sets of finitely generated sets), then the construction of c' can be executed. In this case, we have a means to analyse workflows processing streams, where each stream processor f only has state with finitely many variables taking values in $\mathbb{R}$. The size of the entire stream, however, is not bounded.

3.3.7 Derivative Sensitivity for Row Multiplicities. In this section, we will study the following problem. Let $X = (\mathbb{R}^n, d)$ be a metric space, with d being the ℓ_1-distance (Definition 3.15). Given a function $f : X \to \mathbb{R}$, and the desired level of differential privacy (DP), find a noise distribution $\eta : X \to \mathcal{D}(\mathbb{R})$, such that the probabilistic mapping $g(\vec{x}) := f(\vec{x}) + \eta(\vec{x})$ has that level of DP, and η does not add overly much noise.

In Sec. 3.3.1.2, we discussed that the noise magnitude depends on function sensitivity, i.e. how much a change in the function input affects the function output. There exist different flavours of function sensitivity. The *global* sensitivity is defined for all inputs in the function domain.

Definition 3.36 (global sensitivity)**.** For $f : X \to Y$, the *global sensitivity* of f is

$$GS_f = \max_{x,x' \in X} \frac{d_Y(f(x), f(x'))}{d_X(x, x')} \ .$$

Note that Def. 3.36 is equivalent to Def. 3.5.

Since noise is always added to the output of a query that is applied to a particular state of the database, and some state may require less noise than the other, the noise magnitude may depend on the data to which the function is applied. The *local* sensitivity of a function depends on the actual data instance.

Definition 3.37 (local sensitivity)**.** For $f : X \to Y$, an integer-valued metric $d_X : X \times X \to \mathbb{N}$, and $x \in X$, the *local sensitivity* of f at x is

$$LS_f(x) = \max_{x' \in X : d_X(x,x')=1} d_Y(f(x), f(x')) \ .$$

In this work, we choose local smoothened sensitivity [52] as a suitable basis for choosing the magnitude of the noise. We apply smoothening not to the local sensitivity (although that would work, too), but to its analogue for continuous functions, which we define here and call *derivative sensitivity*.

Derivative sensitivity has to be applied to the semantics of SQL queries. Hence we define a continuous semantics for it, which matches the natural one if applied to databases where records have integer multiplicities. We also show how to compute the smoothened derivative sensitivity of a SQL query.

Let us recall some notions from Sec. 3.3.1. For a set X, let $\mathcal{D}(X)$ denote the set of all probability distributions over it. An element $\chi \in \mathcal{D}(X)$ is a mapping from X to $[0, 1]$, and we use the notation $\chi(x)$ to denote the probability weight that χ assigns to $x \in X$. For distributions $\chi, \chi' \in \mathcal{D}(X)$, their *differential privacy distance* (or *DP-distance*) is defined by

$$d_{\mathrm{dp}}(\chi, \chi') = \sup_{x \in X} \left| \frac{\ln \chi(x)}{\ln \chi'(x)} \right| \ .$$

Given a mapping f from one metric space (X, d_X) to another metric space (Y, d_Y), we say that the *sensitivity* of f is (at most) $c \in \mathbb{R}_+$, if for all $x, x' \in X$, the inequality $d_Y(f(x), f(x')) \leq c \cdot d_X(x, x')$ holds. A mapping $f : X \to \mathcal{D}(Y)$ (where X is a metric space) is *ϵ-differentially private*, if it is ϵ-sensitive from (X, d_X) to $(\mathcal{D}(Y), d_{\mathrm{dp}})$.

In Section 3.3.6, we studied the possibility of distances being not non-negative real numbers, but something more general. One relevant example is the set $\mathcal{F}(\mathbb{R}_+ \times \mathbb{R}_+)$, the set of all *upwards closed* sets of pairs of non-negative real numbers. The upwards closedness means that if $Z \in \mathcal{F}(\mathbb{R}_+ \times \mathbb{R}_+)$, $(\epsilon, \delta) \in Z$, $\epsilon' \geq \epsilon$, and $\delta' \geq \delta$, then $(\epsilon', \delta') \in Z$ as well. One can define ordering and addition on the elements of $\mathcal{F}(\mathbb{R}_+ \times \mathbb{R}_+)$, allowing its elements to be treated as distances and the triangle inequalities to be stated.

This example is significant in defining (ϵ, δ)-differential privacy. Let us recall the definition of the latter.

Definition 3.38 ([36])**.** Let X be a metric space and $f : X \to \mathcal{D}(Y)$. The mapping f is *(ϵ, δ)-differentially private* if for all $Y' \subseteq Y$, and for all x, x', where $d_X(x, x') = 1$, the following inequality holds:

$$\Pr[f(x) \in Y'] \leq e^{\epsilon} \Pr[f(x') \in Y'] + \delta \ . \tag{27}$$

First, if $\chi, \chi' \in \mathcal{D}(X)$, then we define the distance

$$d_{\mathrm{DP}}(\chi, \chi') = \bigcap_{X' \subseteq X} \left\{ (\epsilon, \delta) \;\middle|\; \begin{array}{l} \Pr[x \in X' \,|\, x \leftarrow \chi] \leq e^{\epsilon}(\Pr[x \in X' \,|\, x \leftarrow \chi'] + \delta) \\ \Pr[x \in X' \,|\, x \leftarrow \chi'] \leq e^{\epsilon}(\Pr[x \in X' \,|\, x \leftarrow \chi] + \delta) \end{array} \right\} \tag{28}$$

Clearly, $d_{\mathrm{DP}}(\chi, \chi') \in \mathcal{F}(\mathbb{R}_+ \times \mathbb{R}_+)$. Now, a mapping $f : X \to \mathcal{D}(Y)$ from a metric space X is (ϵ, δ)-differentially private, if it is $\uparrow\{(\epsilon, \delta)\}$-sensitive for the distance d_{DP} being used on $\mathcal{D}(Y)$.

Let us also state how ordering and addition on $\mathcal{F}(\mathbb{R}_+ \times \mathbb{R}_+)$ is defined. Let $Z_1, Z_2 \in \mathcal{F}(\mathbb{R}_+ \times \mathbb{R}_+)$. We have $Z_1 \leq Z_2$ iff $Z_2 \subseteq Z_1$. In this way, the entire set $\mathbb{R}_+ \times \mathbb{R}_+$ is the least element, corresponding to two distributions being equal. Indeed, this equality is expressed by $(0, 0) \in \mathbb{R}_+ \times \mathbb{R}_+$. Such definition of ordering is the standard one. The addition is also standard:

$$Z_1 + Z_2 = \{(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2) \,|\, (\epsilon_1, \delta_1) \in Z_1, (\epsilon_2, \delta_2) \in Z_2\} \ .$$

If Z_1 and Z_2 are upwards closed, then so is $Z_1 + Z_2$ due to the continuousness of $\mathbb{R}$. It is also easy to see that the operation $+$ is associative, commutative, has the zero element $\mathbb{R}_+ \times \mathbb{R}_+$ and is compatible with ordering (meaning that $Z_1 \leq Z_2$ implies $Z_1 + Z_3 \leq Z_2 + Z_3$ for any Z_3).

The following proposition shows that d_{DP} satisfies the triangle inequality.

Proposition 3.52. *Let* $\chi_1, \chi_2, \chi_3 \in \mathcal{D}(X)$. *Then* $d_{\mathrm{DP}}(\chi_1, \chi_3) \leq d_{\mathrm{DP}}(\chi_1, \chi_2) + d_{\mathrm{DP}}(\chi_2, \chi_3)$.

Proof. Let $(\epsilon_1, \delta_1) \in d_{\mathrm{DP}}(\chi_1, \chi_2)$ and $(\epsilon_2, \delta_2) \in d_{\mathrm{DP}}(\chi_2, \chi_3)$. According to the definition of +, the pair $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$ is a member of $d_{\mathrm{DP}}(\chi_1, \chi_2) + d_{\mathrm{DP}}(\chi_2, \chi_3)$. We have to show that it is also a member of $d_{\mathrm{DP}}(\chi_1, \chi_3)$ according to (28). Let $X' \subseteq X$. Then

$$\begin{aligned}\Pr[x \in X' \mid x \leftarrow \chi_1] \leq e^{\epsilon_1}(\Pr[x \in X' \mid x \leftarrow \chi_2] + \delta_1) \leq e^{\epsilon_1}((e^{\epsilon_2}(\Pr[x \in X' \mid x \leftarrow \chi_3] + \delta_2) + \delta_1) = \\ e^{\epsilon_1+\epsilon_2}\Pr[x \in X' \mid x \leftarrow \chi_3] + e^{\epsilon_1+\epsilon_2}\delta_2 + e^{\epsilon_1}\delta_1 \leq e^{\epsilon_1+\epsilon_2}(\Pr[x \in X' \mid x \leftarrow \chi_3] + \delta_2 + \delta_1)\end{aligned}$$

as necessary. □

Let us show the relationship between d_{DP} and (ϵ, δ)-differential privacy. Note that Definition 3.38 is symmetric with respect to x and x', because their role may be swapped.

The formula (27) differs from (28) in one important aspect. Namely, in (28), the quantity δ is multiplied with e^{ϵ}, while in (27), it is not. While the difference of the factor e^{ϵ} seems small in first glance, it is not if we start considering "group privacy", i.e. distances (in X) different from 1. Let $d_X(x, x') = L$. If $f : X \to \mathcal{D}(Y)$ is $\uparrow\{(\epsilon, \delta)\}$-sensitive with respect to the distance d_{DP} on $\mathcal{D}(Y)$, then we know that $(L\epsilon, L\delta) \in d_{\mathrm{DP}}(f(x), f(x'))$. But if f is (ϵ, δ)-differentially private, then we only get

$$\Pr[f(x) \in Y'] \leq e^{L\epsilon}\Pr[f(x') \in Y'] + \frac{e^{L\epsilon} - 1}{e^{\epsilon} - 1}\delta$$

from Definition 3.38.

It is not difficult to show that if we do not multiply δ with e^{ϵ}, then d_{DP} is no longer a distance; in particular, it would not satisfy the triangle inequality. For example, let us pick

$$\chi_1 = \mathit{Ber}(0.01) \quad \chi_2 = \mathit{Ber}(0.03) \quad \chi_3 = \mathit{Ber}(0.07) \quad \epsilon = \ln 2 \quad \delta = 0.01 \ .$$

Here $\mathit{Ber}(p)$ is the *Bernoulli distribution*. It returns 1 with probability p and 0 with probability $1 - p$. We have $(\epsilon, \delta) \in d_{\mathrm{DP}}(\chi_1, \chi_2)$ and also $(\epsilon, \delta) \in d_{\mathrm{DP}}(\chi_2, \chi_3)$, but not $(2\epsilon, 2\delta) \in d_{\mathrm{DP}}(\chi_1, \chi_3)$. Indeed,

$$\begin{aligned}\Pr[x = 1 \mid x \leftarrow \chi_2] = 0.03 = 2 \cdot 0.01 + 0.01 = e^{\epsilon} \cdot \Pr[x = 1 \mid x \leftarrow \chi_1] + \delta \\ \Pr[x = 1 \mid x \leftarrow \chi_3] = 0.07 = 2 \cdot 0.03 + 0.01 = e^{\epsilon} \cdot \Pr[x = 1 \mid x \leftarrow \chi_2] + \delta \\ \Pr[x = 1 \mid x \leftarrow \chi_3] = 0.07 > 4 \cdot 0.01 + 0.02 = e^{2\epsilon} \cdot \Pr[x = 1 \mid x \leftarrow \chi_1] + 2\delta \ .\end{aligned}$$

3.3.7.1 Sensitivity for Continuous Functions. For differentiable functions, the notions of sensitivity and derivatives are very tightly related.

Definition 3.39. Let $f : X \to \mathbb{R}$. The *derivative sensitivity* of f is the following mapping from X to $\mathbb{R}_+$, where $\mathbb{R}_+$ denotes the set of all non-negative real numbers:

$$\mathrm{DS}_f(\vec{x}) = \max_i \left| \frac{\partial f}{\partial x_i}(\vec{x}) \right| \ .$$

Here x_i denotes the i-th component of the vector of variables $\vec{x}$.

For these functions, the derivative sensitivity can be used to obtain results very similar to [52], where local sensitivity was used instead. As we discuss below, derivative sensitivity is usually simpler to use, as long as the mapping f is differentiable.

Definition 3.40 ([52])**.** Let $p : X \to \mathbb{R}$ and $\beta \in \mathbb{R}$. The mapping p is *β-smooth*, if $p(\vec{x}) \leq e^{\beta \cdot d(\vec{x}, \vec{x}')} \cdot p(\vec{x}')$ for all $\vec{x}, \vec{x}' \in X$.

As we see below, crucial in selecting the amount of noise to be added to $f(\vec{x})$ is the knowledge of a β-smooth upper bound on the derivative sensitivity of f. We let c denote such a bound. We consider the same noise distributions as in [52]. For a parameter $\gamma \in \mathbb{R}_+$, $\gamma > 1$, the *generalized Cauchy distribution* $GenCauchy(\gamma) \in \mathcal{D}(\mathbb{R})$ is given by the proportionality

$$GenCauchy(\gamma)(x) \propto \frac{1}{1+|x|^\gamma}$$

("usual" Cauchy distribution is obtained for $\gamma = 2$). Noise distributed by generalized Cauchy distribution, weighed by a smooth upper bound on the derivative sensitivity of f, allows us to achieve ϵ-DP.

Theorem 3.53. *Let $\gamma, b, \beta \in \mathbb{R}_+$, $\gamma > 1$. Let $\epsilon = (\gamma+1)(b+\beta)$. Let η be a random variable distributed according to GenCauchy(γ). Let c be a β-smooth upper bound on DS_f for a function $f : X \to \mathbb{R}$. Then $g(\vec{x}) : f(\vec{x}) + \frac{c(\vec{x})}{b} \cdot \eta$ is ϵ-differentially private.*

Proof. Let $\eta \sim GenCauchy(\gamma)$. The generalized Cauchy distribution is relatively stable under shifts and stretchings, satisfying the following inequalities for all $a_1, a_2, c_1, c_2 \in \mathbb{R}$, which we state here without proof [52]:

$$d_{\mathrm{dp}}(a_1 + c_1 \cdot \eta, a_2 + c_1 \cdot \eta) \le (\gamma+1) \cdot \left| \frac{a_2 - a_1}{c_1} \right|$$
$$d_{\mathrm{dp}}(c_1 \cdot \eta, c_2 \cdot \eta) \le (\gamma+1) \cdot \left| \ln \frac{c_2}{c_1} \right| \quad .$$

The combination of these two inequalities gives

$$d_{\mathrm{dp}}(a_1 + c_1 \cdot \eta, a_2 + c_2 \cdot \eta) \le (\gamma+1) \cdot \left(\frac{|a_2 - a_1|}{\max\{|c_1|, |c_2|\}} + \left| \ln \frac{c_2}{c_1} \right| \right) \quad . \tag{29}$$

Let $\vec{x}, \vec{x'} \in X$. Suppose that they differ only in the i_0-th coordinate. W.l.o.g. assume that $x'_{i_0} \ge x_{i_0}$. Denote $L = x'_{i_0} - x_{i_0}$. We have to show that $d_{\mathrm{dp}}(g(\vec{x'}), g(\vec{x})) \le \epsilon L = (\gamma+1)(b+\beta)L$.

We can substitute the definition of g into the left side of the desired inequality above, and using the inequality (29) and the definition of smoothness, obtain

$$d_{\mathrm{dp}}(g(\vec{x}), g(\vec{x'})) = d_{\mathrm{dp}}(f(\vec{x}) + \frac{c(\vec{x})}{b} \cdot \eta, f(\vec{x'}) + \frac{c(\vec{x'})}{b} \cdot \eta) \le$$
$$(\gamma+1) \cdot \left(b \cdot \frac{|f(\vec{x'}) - f(\vec{x})|}{|c(\vec{x})|} + \left| \ln \frac{c(\vec{x'})}{c(\vec{x})} \right| \right) \le (\gamma+1) \cdot \left(b \cdot \frac{|f(\vec{x'}) - f(\vec{x})|}{|c(\vec{x})|} + \beta L \right)$$

Unfortunately, we cannot directly bound $|f(\vec{x'}) - f(\vec{x})|/|c(\vec{x})|$ with L. Instead, if we let $\vec{y}[v]$ denote the tuple $\vec{x}$, where the i_0-th component is replaced with v, then we can only claim (using the mean value theorem), that there exists some v_0 in the segment (x_{i_0}, x'_{i_0}) satisfying

$$|f(\vec{x'}) - f(\vec{x})| = \left| \frac{\partial f}{\partial x_{i_0}}(\vec{y}[v_0]) \right| \cdot (x'_{i_0} - x_{i_0}) \le |c(\vec{y}[v_0])| \cdot L,$$

where the last inequality is due to c being an upper bound on the derivative sensitivity of f. However, by using this claim many times, we obtain the necessary inequality as follows. Let $n \in \mathbb{N}$ be arbitrary. Let $v_0 = x_{i_0}$, $v_n = x'_{i_0}$ and $v_i = ((n-i) \cdot x_{i_0} + i \cdot x'_{i_0})/n$, i.e. the values $v_0, \ldots, v_n$ are evenly distributed from x_{i_0} to x'_{i_0}. Again, the mean value theorem implies that there exist $t_1, \ldots, t_n$ with $v_{i-1} \le t_i \le v_i$, satisfying

$$|f(\vec{y}[v_i]) - f(\vec{y}[v_{i-1}])| = \left| \frac{\partial f}{\partial x_{i_0}}(\vec{y}[t_i]) \right| \cdot (v_i - v_{i-1}) \le |c(\vec{y}[t_i])| \cdot \frac{L}{n} \le e^{\beta L/n} \cdot |c(\vec{y}[v_{i-1}])| \cdot \frac{L}{n}$$

for all $i \in \{1, \ldots, n\}$. Here the last inequality follows from the β-smoothness of c. We can use these claims together with the triangle inequality and obtain

$$d_{\mathrm{dp}}(g(\vec{x}), g(\vec{x}')) \leq \sum_{i=1}^{n} d_{\mathrm{dp}}(g(\vec{y}[v_{i-1}]), g(\vec{y}[v_i])) = \sum_{i=1}^{n} d_{\mathrm{dp}}(f(\vec{y}[v_{i-1}]) + \frac{c(\vec{y}[v_{i-1}])}{b} \cdot \eta, f(\vec{y}[v_i]) + \frac{c(\vec{y}[v_i])}{b} \cdot \eta) \leq$$

$$(\gamma+1) \sum_{i=1}^{n} \left(b \cdot \frac{|f(\vec{y}[v_i]) - f(\vec{y}[v_{i-1}])|}{|c(\vec{y}[v_{i-1}])|} + \frac{\beta L}{n} \right) \leq (\gamma+1) \sum_{i=1}^{n} \left(b \cdot e^{\beta L/n} \cdot \frac{L}{n} + \frac{\beta L}{n} \right) = (\gamma+1)(be^{\beta L/n} + \beta)L \ .$$

This inequality holds for any $n \in \mathbb{N}$. If $n \to \infty$ then $e^{\beta L/n} \to 1$ and we obtain the inequality that we had to show.

If $\vec{x}$ and $\vec{x}'$ differ in more than one coordinate, then we can transform $\vec{x}$ to $\vec{x}'$ by changing one coordinate at a time, and using the triangle inequality. □

Generalized Cauchy distributions have heavy tails, meaning that using the mechanism of Theorem 3.53 will in general introduce a lot of noise. Less noise is possible if we resort to (ϵ, δ)-DP. In this case, we may add noise from the usual Laplacian distribution, which has light tails. The following theorem makes use of the distribution $Lap(1) \in \mathcal{D}(\mathbb{R})$, defined by $Lap(1)(x) \propto e^{-|x|}$. We first have to state the results about the self-similarity of $Lap(1)$ under shifting and stretching.

Lemma 3.54. *Let $\eta \sim Lap(1)$. Let $a_1, a_2 \in \mathbb{R}$, $c_1, c_2 \in \mathbb{R}_+$, $c_1 \leq c_2$. Define $\beta = \ln(c_2/c_1)$ and let $\epsilon \geq \beta$. Let $\delta \geq e^{-\epsilon-(\epsilon+\beta)/(e^{\beta}-1)}$. Then the following holds.*

$$\left(\frac{|a_2 - a_1|}{c_1}, 0 \right) \in d_{\mathrm{DP}}(a_1 + c_1 \cdot \eta, a_2 + c_1 \cdot \eta)$$
$$(\epsilon, \delta) \in d_{\mathrm{DP}}(c_1 \cdot \eta, c_2 \cdot \eta) \ .$$

Proof. The probability density functions (PDF) and the cumulative density functions (CDF) of the distributions named above are the following:

$$\mathrm{PDF}_{c_1 \cdot \eta}(x) = \frac{1}{2c_1} e^{-|x|/c_1} \qquad \mathrm{PDF}_{a_1 + c_1 \cdot \eta}(x) = \frac{1}{2c_1} e^{-|x-a_1|/c_1}$$
$$\mathrm{PDF}_{c_2 \cdot \eta}(x) = \frac{1}{2c_2} e^{-|x|/c_2} \qquad \mathrm{PDF}_{a_2 + c_1 \cdot \eta}(x) = \frac{1}{2c_1} e^{-|x-a_2|/c_1}$$

and

$$\mathrm{CDF}_{c_1 \cdot \eta}(x) = \begin{cases} e^{x/c_1}/2, & \text{if} x < 0 \\ 1 - e^{-x/c_1}/2, & \text{if} x \geq 0 \end{cases} \qquad \mathrm{CDF}_{c_2 \cdot \eta}(x) = \begin{cases} e^{x/c_2}/2, & \text{if} x < 0 \\ 1 - e^{-x/c_2}/2, & \text{if} x \geq 0 \end{cases}$$

The first claim of the lemma is shown by

$$\max_{x \in \mathbb{R}} \left| \ln \frac{\mathrm{PDF}_{a_1 + c_1 \cdot \eta}(x)}{\mathrm{PDF}_{a_2 + c_1 \cdot \eta}(x)} \right| = \max_{x \in \mathbb{R}} \left| \ln \frac{e^{-|x-a_1|/c_1}}{e^{-|x-a_2|/c_1}} \right| \leq \ln e^{|a_1 - a_2|/c_1} = \frac{|a_2 - a_1|}{c_1},$$

showing that $\frac{|a_2-a_1|}{c_1} \geq d_{\mathrm{dp}}(a_1 + c_1 \cdot \eta, a_2 + c_1 \cdot \eta)$. To show the second claim, consider the following function f:

$$f(x) = \left| \ln \frac{\mathrm{PDF}_{c_1 \cdot \eta}(x)}{\mathrm{PDF}_{c_2 \cdot \eta}(x)} \right|.$$

We are interested in the set of x-s that satisfy $f(x) \leq \epsilon$. We have

$$f(x) = \left| \ln \left(\frac{c_2}{c_1} \cdot e^{|x|/c_2 - |x|/c_1} \right) \right| = \left| \beta - \frac{c_2 - c_1}{c_1 c_2} |x| \right| \ .$$

The condition $f(x) \leq \epsilon$ is equivalent to

$$|x| \leq (\epsilon + \beta)\frac{c_1 c_2}{c_2 - c_1} \quad . \tag{30}$$

To obtain the distance $\uparrow\{(\epsilon, \delta)\}$, it is sufficient to take δ equal to $e^{-\epsilon}$ times the probability that either x, when sampled according to either $c_2 \cdot \eta$ or $c_1 \cdot \eta$, does not satisfy (30). This probability is larger for $c_2 \cdot \eta$ because $c_2 \geq c_1$. Let us compute this probability.

$$\Pr[x < 0 \wedge f(x) > \epsilon \,|\, x \leftarrow c_2 \cdot \eta] = \frac{1}{2} e^{(\epsilon+\beta)\frac{c_1}{c_2 - c_1}} = \frac{1}{2} e^{(\epsilon+\beta)\frac{1}{e^\beta - 1}} \quad .$$

The probability that we are looking for is twice the quantity above. Multiplying it with $e^{-\epsilon}$ gives us the statement of the lemma. □

The next lemma provides a more coarse, but simpler upper bound for the DP-distance between stretched versions of the Laplace distribution.

Lemma 3.55. *Let $\eta \sim Lap(1)$. Let $c_1, c_2 \in \mathbb{R}_+$, $c_1 \leq c_2$. Define $\beta = \ln(c_2/c_1)$. Let $\epsilon \geq \beta$. Let $k = 1 + \epsilon/\beta$. Then $(\epsilon, e^{-k}) \in d_{\mathrm{DP}}(c_1 \cdot \eta, c_2 \cdot \eta)$.*

Proof. Let $\delta = e^{-\epsilon - \frac{\epsilon+\beta}{e^\beta - 1}}$. By the previous lemma, $(\epsilon, \delta) \in d_{\mathrm{DP}}(c_1 \cdot \eta, c_2 \cdot \eta)$. We will now show that $e^{-k} \geq \delta$.

Indeed,

$$\delta \leq e^{-k} \Leftrightarrow e^{-\epsilon - \frac{\epsilon+\beta}{e^\beta - 1}} \leq e^{-k} \Leftrightarrow -\epsilon - \frac{\epsilon + \beta}{e^\beta - 1} \leq -k \Leftrightarrow (k-1)\beta + \frac{k\beta}{e^\beta - 1} \geq k \Leftrightarrow \frac{1}{e^\beta - 1} \geq \frac{1}{\beta} - \frac{k-1}{k} \Leftarrow$$
$$\frac{1}{e^\beta - 1} \geq \frac{1}{\beta} - \frac{1}{2} \Leftrightarrow \frac{1}{\beta} \leq \frac{1}{2} + \frac{1}{e^\beta - 1} \Leftrightarrow \beta \geq \frac{2(e^\beta - 1)}{e^\beta + 1} \Leftrightarrow \beta + \frac{4}{e^\beta + 1} \geq 2,$$

where the "⇐" claim holds because $k \geq 2$. Consider now the function $f(x) = x + 4/(e^x + 1)$. We have $f(0) = 2$. Also, f is a monotone function. Hence the claim $\beta + 4/(e^\beta + 1) \geq 2$ holds. □

We can now proceed to state the differential privacy theorem for the added Laplace noise. We start with a technical notion that we use in the proof, and which can also be used to state the additional conditions when we get a tighter bound for the (ϵ, δ)-differential privacy of the noised function.

Definition 3.41. A *path* in X is a continuous function $h : [0, 1] \to X$. The path h is *shortest*, if for all $x_1, x_2, x_3 \in [0, 1]$, $x_1 \leq x_2 \leq x_3$, the equality $d_X(h(x_1), h(x_3)) = d_X(h(x_1), h(x_2)) + d_X(h(x_2), h(x_3))$ holds.

Theorem 3.56. *Let $b, \beta, \epsilon \in \mathbb{R}_+$, $b > 0$, $b + \beta \leq \epsilon$. Define $k = 1 + (\epsilon - b)/\beta$. Let $\delta = e^{-k}$. Let η be a random variable distributed according to Lap(1). Let c be a β-smooth upper bound on DS_f for a function $f : X \to \mathbb{R}$. Define $g(\vec{x}) := f(\vec{x}) + \frac{c(\vec{x})}{b} \cdot \eta$. Then*

- *for any $\vec{x}_1, \vec{x}_2 \in X$, $(\epsilon \cdot L, 2\delta) \in d_{\mathrm{DP}}(g(\vec{x}_1), g(\vec{x}_2))$, where $L = d_X(\vec{x}_1, \vec{x}_2)$;*
- *(in particular,) g is $(\epsilon, 2e^\epsilon\delta)$-differentially private.*

If, additionally for any two points $\vec{x}_1, \vec{x}_2 \in X$ there exists a shortest path h in X, such that c is monotonic along that path, then the factor "2" in previous statements can be removed..

Proof. Let b, β, ϵ be as in the statement of the theorem. Let $\eta \sim Lap(1)$ and $\vec{x}_1, \vec{x}_2 \in X$. Let $L = d_X(\vec{x}_1, \vec{x}_2)$. Let h be a shortest path from $\vec{x}_1$ to $\vec{x}_2$. Let $\vec{x}_\mu$ be a point on the path h, such that $c(\vec{x}_\mu) = \max_{t \in [0,1]} c(h(t))$. Let $L_1 = d_X(\vec{x}_1, \vec{x}_\mu)$ and $L_2 = d_X(\vec{x}_\mu, \vec{x}_2)$. Note that $L = L_1 + L_2$. Define the following probability distributions:

$$\chi_1 = f(\vec{x}_1) + \frac{c(\vec{x}_1)}{b} \cdot \eta \qquad \chi_2 = f(\vec{x}_1) + \frac{c(\vec{x}_\mu)}{b} \cdot \eta$$
$$\chi_3 = f(\vec{x}_2) + \frac{c(\vec{x}_\mu)}{b} \cdot \eta \qquad \chi_4 = f(\vec{x}_2) + \frac{c(\vec{x}_2)}{b} \cdot \eta \quad .$$

We want to show that $(\epsilon L, 2\delta) \in d_{\mathrm{DP}}(\chi_1, \chi_4)$.

By Lemma 3.54, $(b \cdot |f(\vec{x}_2) - f(\vec{x}_1)|/c(\vec{x}_\mu), 0) \in d_{\mathrm{DP}}(\chi_2, \chi_3)$. The difference between $f(\vec{x}_2)$ and $f(\vec{x}_1)$ can be upper-bounded as follows:

$$|f(\vec{x}_2) - f(\vec{x}_1)| = \left| \int_h f'(h)\, ds \right| \leq \left| \int_0^1 f'(h(t))\|h'(t)\|\, dt \right| \leq \left| \int_0^1 c(h(t))\|h'(t)\|\, dt \right| \leq$$
$$\int_0^1 c(\vec{x}_\mu)\|h'(t)\|\, dt = c(\vec{x}_\mu) \cdot d_X(h(1) - h(0)) = L \cdot c(\vec{x}_\mu),$$

hence $(bL, 0) \in d_{\mathrm{DP}}(\chi_2, \chi_3)$. Note that, in $\int_h f'(h)\, ds$, the integral is over the path h, and the derivative f' is w.r.t. distance d_X along the path h. This derivative exists for a differentiable f, which follows from the definition of DS_f. More generally, if X is a *Banach space*, its existence of follows from the existence of the Fréchet derivative of f (we will discuss Banach spaces and Fréchet derivatives in more details in Sec. 3.3.8.2). The ds is an infinitesimal distance (according to d_X) along the path h, and the h in $f'(h)$ denotes the point on the path h.

We will now compare χ_1 and χ_2. We use the previous lemma with the following instantiations:

Quantity in Lemma 3.55	Instantiation
c_1	$c(\vec{x}_1)$
c_2	$c(\vec{x}_\mu)$
β	$\ln(c(\vec{x}_\mu)/c(\vec{x}_1))$
ϵ	$(\epsilon - b)L_1$
k	$1 + (\epsilon - b)L_1 / \ln(c(\vec{x}_\mu)/c(\vec{x}_1))$

Lemma 3.55 required that $\epsilon \geq \beta$. This condition is translated to

$$(\epsilon - b)L_1 \geq \ln \frac{c(\vec{x}_\mu)}{c(\vec{x}_1)} \ .$$

Let us verify that it holds:

$$\ln \frac{c(\vec{x}_\mu)}{c(\vec{x}_1))} \leq \beta \cdot d_X(\vec{x}_1, \vec{x}_\mu) \leq \beta L_1 \leq (\epsilon - b)L_1 \qquad \blacksquare \ .$$

We also lower-bound the value of k from Lemma 3.55, in order to simplify its expression:

$$1 + \frac{(\epsilon - b)L_1}{\ln \frac{c(\vec{x}_\mu)}{c(\vec{x}_1))}} \geq 1 + \frac{(\epsilon - b)L_1}{\beta L_1} = k \ .$$

We obtain

$$((\epsilon - b)L_1, e^{-k}) \in d_{\mathrm{DP}}(\chi_1, \chi_2) \ .$$

Similarly, we can obtain

$$((\epsilon - b)L_2, e^{-k}) \in d_{\mathrm{DP}}(\chi_3, \chi_4) \ .$$

Using the triangle inequality, we can combine $d_{\mathrm{DP}}(\chi_i, \chi_{i+1})$ for $i \in \{1, 2, 3\}$:

$$(\epsilon L, 2e^{-k}) \in d_{\mathrm{DP}}(\chi_1, \chi_4)$$

as required. If $d_X(\vec{x}_1, \vec{x}_2) = 1$, then $(\epsilon, 2e^{-k}) \in d_{\mathrm{DP}}(g(\vec{x}_1), g(\vec{x}_2))$, i.e. g is $(\epsilon, 2\delta)$-differentially private.

If c is monotone along the path h, then the point $\vec{x}_\mu$ coincides with either $\vec{x}_1$ or $\vec{x}_2$. W.l.o.g. assume $\vec{x}_\mu = \vec{x}_1$. Then $\chi_1 = \chi_2$ and $(0, 0) \in d_{\mathrm{DP}}(\chi_1, \chi_2)$. The triangle inequality now gives $(\epsilon L, e^{-k}) \in d_{\mathrm{DP}}(\chi_1, \chi_4)$. □

Even though we obtained a result about the (ϵ, δ)-differential privacy level of g, the guarantee given by Theorem 3.56 is of a rather different kind. Indeed, it gives a reasonable "group privacy" also when the arguments to f are far apart.

$$E ::= T \mid \pi_S(E) \mid \sigma_\theta(E) \mid E \times E \mid E \cup E \mid E \cap E \mid E - E \mid \rho_{S \to S'}(E) \mid \mathsf{Dis}(E)$$

$$\theta ::= 0 \mid 1 \mid P(\vec{s}) \mid \theta \wedge \theta \mid \neg\theta$$

Here $E \in \mathbf{RExp}$, $\theta \in \mathbf{BExp}$, $T \in \mathsf{R}$, $S, S' \in \mathsf{N}^{*\neq}$ and $|S| = |S'|$, $P \in \mathsf{P}$, $\vec{s} \in \mathsf{N}^*$.

Figure 37: Syntax of Relational Algebra

3.3.7.2 Continuous Semantics for Relational Algebra Queries. SQL is used to specify queries against relational databases, i.e. databases made up of several tables, where each table is a multiset of records of certain type. We are going to apply the theory described above to the differential privacy of SQL queries. For this to be possible, we need to give a semantics for SQL that is based on the multiplicities of records being real numbers. When applied to "normal" databases, i.e. to databases where each row in each table has an integer multiplicity, then our semantics will coincide with the standard semantics of SQL.

The semantics of SQL queries has recently been studied in [53], where its expressivity has been shown to be the same as a relational algebra fragment. The fragment consists of a following components:

Names. These are used to name the columns of tables/relations. Let N be the set of all names. This set is assumed to be countably infinite.

Values. These inhabit the cells in relations. To simplify the treatment, all values are taken to have the same type. Let C be the set of all values.

Basic relations. These are the names of the tables in the database. Let R be the (finite) set of these names.

Predicate symbols. These are used to state boolean conditions on top of values. Let P be the set of these symbols. We assume that a symbol for equality checking is an element of P.

On top of these sets, the syntax of relational algebra is built. The syntax defines two categories: relational algebra expressions (the set **RExp**) and boolean expressions (the set **BExp**) given in Fig. 37. Given a set X, we let X^* denote all sequences of the elements of X. We also let $X^{*\neq} \subset X^*$ denote all such sequences where all elements are different. If $Y \in X^{*\neq}$, then we may also interpret Y as a subset of X, consisting of all elements that occur in it.

We see that a relational algebra expression can be a table in the database, a projection, a filter, a combination of results of two expressions (cartesian product, union, intersection, difference), renaming of columns. Finally, if E is an expression then $\mathsf{Dis}(E)$ is also an expression, corresponding to the removal of repeated rows from the result of E. It corresponds to the `SELECT DISTINCT` keyword in SQL.

The semantics of relational algebra is defined as follows. For each predicate symbol $P \in \mathsf{P}$, let there be an associated semantic function $[\![P]\!] : \mathsf{C}^* \to \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$. Let there be a mapping $\ell : \mathsf{R} \to \mathsf{N}^{*\neq}$, giving the schema of each basic relation. For a schema $S \in \mathsf{N}^{*\neq}$, the set of relations matching S is

$$\mathbf{Rels}(S) = \{\mathbf{r} : (S \to \mathsf{C}) \to \mathbb{R}_+ \mid |\mathrm{supp}\ \mathbf{r}| < \infty\},$$

where $\mathrm{supp}\ \mathbf{r} \subseteq (S \to \mathsf{C})$ denotes the *support* of $\mathbf{r}$ — the set of all such tuples $X : S \to \mathsf{C}$ where $\mathbf{r}(X) \neq 0$. We see that $\mathbf{r}$ gives the multiplicity of each possible row in a table with schema S, and the multiplicities can be fractional numbers. A database D maps each table name $T \in \mathsf{R}$ to a relation $D^T \in \mathbf{Rels}(\ell(T))$.

We extend the schema ℓ to all relational algebra expressions in the manner given in Figure 38. In this extension, the *type* $\tau(\theta) \subset \mathsf{N}$ of a boolean expression θ is the set of all names that occur in it. We consider a relational algebra expression E syntactically valid only if $\ell(E)$ is defined, and define the semantics only for syntactically valid expressions.

To define the semantics of removing repeated rows, we have to explain what happens with their multiplicity. We do not have to fix this in a unique manner, but can choose the behavior of the `DISTINCT`-keyword in a way that is most beneficial to our analysis, given that it behaves in the traditional way on

$$
\begin{aligned}
\ell(\pi_S(E)) &= S && \text{undefined, if} S \not\subseteq \ell(E)\\
\ell(\sigma_\theta(E)) &= \ell(E) && \text{undefined, if} \tau(\theta) \not\subseteq \ell(E)\\
\ell(E_1 \times E_2) &= \ell(E_1) \| \ell(E_2) && \text{undefined, if} \ell(E_1) \cap \ell(E_2) \neq \emptyset\\
\ell(E_1 \cup E_2) &= \ell(E_1) && \text{undefined, if} \ell(E_1) \neq \ell(E_2)\\
\ell(E_1 \cap E_2) &= \ell(E_1) && \text{undefined, if} \ell(E_1) \neq \ell(E_2)\\
\ell(E_1 - E_2) &= \ell(E_1) && \text{undefined, if} \ell(E_1) \neq \ell(E_2)\\
\ell(\rho_{S \to S'}(E)) &= S' && \text{undefined, if} \ell(E) \neq S\\
\ell(\mathsf{Dis}(E)) &= \ell(E)
\end{aligned}
$$

Figure 38: Schemas of Relational Expressions

$$
\begin{aligned}
[\![0]\!](\mathsf{r}) &= 0\\
[\![1]\!](\mathsf{r}) &= 1\\
[\![P(s_1, \ldots, s_k)]\!](\mathsf{r}) &= [\![P]\!](\mathsf{r}(s_1), \ldots, \mathsf{r}(s_k))\\
[\![\theta_1 \wedge \theta_2]\!](\mathsf{r}) &= \min\{[\![\theta_1]\!](\mathsf{r}), [\![\theta_2]\!](\mathsf{r})\}\\
[\![\neg\theta]\!](\mathsf{r}) &= 1 - [\![\theta]\!](\mathsf{r})
\end{aligned}
$$

Figure 39: Semantics of Boolean Expressions

$$
\begin{aligned}
[\![T]\!]_D(\mathsf{r}) &= D^T(\mathsf{r})\\
[\![\pi_S(E)]\!]_D(\mathsf{r}) &= \sum_{\substack{\mathsf{r}' : \ell(E) \to \mathsf{C}\\ \forall s \in S : \mathsf{r}(s) = \mathsf{r}'(s)}} [\![E]\!]_D(\mathsf{r}')\\
[\![\sigma_\theta(E)]\!]_D(\mathsf{r}) &= [\![E]\!]_D(\mathsf{r}) \cdot [\![\theta]\!](\mathsf{r})\\
[\![E_1 \times E_2]\!]_D(\mathsf{r}) &= [\![E_1]\!]_D(\mathsf{r}|_{\ell(E_1)}) \cdot [\![E_2]\!]_D(\mathsf{r}|_{\ell(E_2)})\\
[\![E_1 \cup E_2]\!]_D(\mathsf{r}) &= [\![E_1]\!]_D(\mathsf{r}) + [\![E_2]\!]_D(\mathsf{r})\\
[\![E_1 \cap E_2]\!]_D(\mathsf{r}) &= \min\{[\![E_1]\!]_D(\mathsf{r}), [\![E_2]\!]_D(\mathsf{r})\}\\
[\![E_1 - E_2]\!]_D(\mathsf{r}) &= \max\{0, [\![E_1]\!]_D(\mathsf{r}) - [\![E_2]\!]_D(\mathsf{r})\}\\
[\![\rho_{s_1 \cdots s_k \to s'_1 \cdots s'_k}(E)]\!]_D(\mathsf{r}) &= [\![E]\!]_D(\{\forall i : s_i \mapsto \mathsf{r}(s'_i)\})\\
[\![\mathsf{Dis}(E)]\!]_D(\mathsf{r}) &= \mathsf{Sgm}([\![E]\!]_D(\mathsf{r}))
\end{aligned}
$$

Figure 40: Semantics of Expressions of Relational Algebra

integral multiplicities. Hence we state that there is a function $\mathsf{Sgm} : \mathbb{R}_+ \to \mathbb{R}_+$, which is continuous, differentiable, and *monotonic*. Moreover, it satisfies $\mathsf{Sgm}(0) = 0$ and $\mathsf{Sgm}(x) = 1$ for all $x \geq 1$.

The semantics $[\![\theta]\!]$ of a boolean expression θ is a mapping from a row (with the type $S \to C$, where $S \supseteq \tau(\theta)$) to a boolean value. It is given in Figure 39.

The semantics $[\![E]\!]_D$ (relative to a database D) of a relational algebra expression E is a relation in **Rels**$(\ell(E))$, i.e. a mapping from rows to their multiplicities, where a row r has the type $\ell(E) \to \mathsf{C}$. It is given in Figure 40.

The purpose of adding the noise to the result of a query E is to hide small changes in the database that is the subject of the query. Hence the semantics of E has to be seen as a function that takes a

database as an input. We have

$$[\![E]\!] : \prod_{T \in \mathsf{R}} \mathbf{Rels}(\ell(T)) \to \mathbf{Rels}(\ell(E)),$$

where the domain of this function is the type of databases. The notation $[\![E]\!]_D$ means the application of the function $[\![E]\!]$ to the database D. We consider $[\![E]\!]$ as a multivariate function (also returning many values), with each variable taking values in $\mathbb{R}_+$. A variable corresponds to a possible row in one of the tables, i.e. it is identified by

1. the name of the table $T \in \mathsf{R}$;
2. the values in the row: $\mathbf{r} : \ell(T) \to \mathsf{C}$.

Let $x_{T;\mathbf{r}}$ be the variable that corresponds to the potential row $\mathbf{r}$ in the table T. When describing a database, the value of $x_{T;\mathbf{r}}$ is the multiplicity of the row $\mathbf{r}$ in the table T.

SQL rewriting. There may be a number of different ways to extend an integral semantics of a SQL query to a fractional one. All these fractional extensions are equally valid as the basis for computing the magnitude of noise to be added to the query result in order to obtain a certain level of differential privacy. They may lead to the selection of different amounts of noise, or its addition in different places, resulting in different accuracy, but the privacy guarantee remains, because all different fractional semantics agree on integral points.

A database engine typically rewrites SQL queries in order to make their execution more efficient. The engine aims to preserve the integral semantics of the query, but does not care about the fractional semantics. Hence the fractional semantics may change during rewriting. Nevertheless, the discussion in the previous paragraph shows that the privacy analysis performed on the initial query remains valid also for the rewritten query. Our extension of the semantics hence does not bring about a need to change the query execution strategies of database engines.

3.3.7.3 Smooth Derivative Sensitivity for Relational Algebra Queries. A relational algebra expression returns relations, but our noise addition mechanisms naturally work with functions that return a numeric value. We note that this restriction is similar to other differential privacy mechanisms for database queries, e.g. PINQ [54]. These numbers could be the values of particular cells in the returned relation, or some summary of the relation. Inspired by the SQL-statements turning up in scenarios investigated by Brandeis CRTs, we focus on counting the rows of a relation. Let

$$\mathsf{Count}_E(D) = [\![\pi_\emptyset(E)]\!]_D(\emptyset)$$

be the number of rows in the result of the query E (as a function of the database instance D). Let the derivative sensitivity of this be

$$\mathsf{DSC}_E(D) = \mathrm{DS}_{\mathsf{Count}_E}(D)$$

Our methods for finding smooth derivative sensitivity of Count_E-queries cannot currently handle all of the relational algebra specified in Figure 37, but they can cover expressions E corresponding to the SQL-queries of the form

```
SELECT A1,...,Ak FROM T1,...,Tn WHERE condition        (31)
```

where *condition* may be any boolean expression in Figure 37 and where the tables may appear in the list `T1,...,Tn` several times.

Finding the derivative. As defined previously, Let $x_{T;\mathbf{r}}$ is the multiplicity of the row $\mathbf{r}$ in the table T of the database D. Considering different cases of E, we find the derivative of the count function with respect to $x_{T;\mathbf{r}}$. The simplest case is, when E is just a table. Then

$$\frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{T'}(D)) = \begin{cases} 1, & \text{if} T = T' \\ 0, & \text{if} T \neq T' \end{cases}.$$

The derivative of the count of the projection, renaming, and filtering operations are trivial to inductively specify:

$$\frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\pi_S(E)}(D)) = \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_E(D))$$
$$\frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\rho_{s_1\cdots s_k\to s'_1\cdots s'_k}(E)}(D)) = \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_E(D)) \ .$$

To compute the derivative of the count of a filtering operation, we extend the definition of $[\![\theta]\!]$ so that it can also be applied to rows that do not contain all the columns referenced in θ. In this case, the unbound variables are assumed to be existentially quantified, i.e.

$$[\![\theta]\!](\mathbf{r}) = \sup_{\mathbf{r}' \text{ extends } \mathbf{r}} [\![\theta]\!](\mathbf{r}'),$$

where $\mathbf{r}' : \tau(\theta) \to \mathsf{C}$ ranges over all possible rows that define the attributes used by θ. The derivative of the count of the filtering is the derivative of the count of the original expression, moderated by the truth value of θ on the row $\mathbf{r}$.

$$\frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\sigma_\theta(T)}(D)) = \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_T(D)) \cdot [\![\theta]\!](\mathbf{r}) \ .$$

The derivative of the count of a Cartesian product is defined as the derivative of a product function:

$$\frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{E_1\times E_2}(D)) = \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{E_1}(D)) \cdot \mathsf{Count}_{E_2}(D) + \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{E_2}(D)) \cdot \mathsf{Count}_{E_1}(D)$$

We are interested in the case where filtering is applied to the product (and projection is applied afterwards, but its derivative was trivial). In this case, the derivative can be upper-bounded as follows:

$$\begin{aligned}\frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\sigma_\theta(E_1\times E_2)}(D)) \le \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\sigma_\theta(E_1)\times\sigma_\theta(E_2)}(D)) = \\ \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\sigma_\theta(E_1)}(D)) \cdot \mathsf{Count}_{\sigma_\theta(E_2)}(D) + \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\sigma_\theta(E_2)}(D)) \cdot \mathsf{Count}_{\sigma_\theta(E_1)}(D) \le \\ \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{E_1}(D)) \cdot \mathsf{Count}_{\sigma_\theta(E_2)}(D) + \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{E_2}(D)) \cdot \mathsf{Count}_{\sigma_\theta(E_1)}(D)\end{aligned}$$

If the Cartesian product is applied to any number of expressions E_j, we get

$$\frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\prod_j E_j}(D)) = \sum_i \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{E_i}(D)) \cdot \mathsf{Count}_{\prod_{j\neq i} E_j}(D),$$

which can be upper-bounded as

$$\begin{aligned}&\frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\sigma_\theta(\prod_j E_j)}(D)) \le \\ &\sum_i \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\sigma_\theta(E_i)}(D)) \cdot \mathsf{Count}_{\sigma_\theta(\prod_{j\neq i} E_j)}(D) \le \sum_i \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{E_i}(D)) \cdot \mathsf{Count}_{\sigma_\theta(\prod_{j\neq i} E_j)}(D) \quad (32)\end{aligned}$$

If each subquery E_i is a table ($E_i = T_i$) and all T_i are distinct then only one of the summands can be nonzero—the one where $T = T_i$. If the T_i are not distinct then there would be one summand for each copy of the table T:

$$\begin{aligned}&\frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\sigma_\theta(\prod_j T_j)}(D)) \le \\ &\qquad \sum_{i:\ T_i=T} \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{T_i}(D)) \cdot \mathsf{Count}_{\sigma_{\theta[\mathbf{r}/T_i]}(\prod_{j\neq i} T_j)}(D) = \sum_{i:\ T_i=T} \mathsf{Count}_{\sigma_{\theta[\mathbf{r}/T_i]}(\prod_{j\neq i} T_j)}(D) \ .\end{aligned}$$

Here we can be more precise than in the computation (32). As we know that $\mathbf{r}$ corresponds to a row in the relational algebra expression T_i, we can specialize the filter θ, making the references to the columns of T_i to be equal to the values of the attributes in $\mathbf{r}$. We denote the specialization with $\theta[\mathbf{r}/T_i]$. It does not quite fit the syntax in Figure 37, but its semantics can be straightforwardly defined.

Finding the derivative sensitivity. Now the derivative sensitivity is found by taking the maximum of all the derivatives (which are non-negative for the current case, so it is not necessary to take the absolute value).

$$\mathsf{DSC}_{\sigma_\theta(\prod_j T_j)}(D) = \max_{T,\mathbf{r}} \frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\sigma_\theta(\prod_j T_j)}(D))$$

$$\mathsf{DSC}_{\sigma_\theta(\prod_j T_j)}(D) \le \max_{T,\mathbf{r}} \sum_{i:\ T_i=T} \mathsf{Count}_{\sigma_{\theta[\mathbf{r}/T_i]}(\prod_{j\neq i} T_j)}(D)$$

Patterns. Above we showed how to compute an upper bound of $\mathsf{Count}_{\sigma_{\theta[\mathbf{r}/T_i]}(\prod_{j\neq i} T_j)}(D)$ for each possible row r separately. However, the number of possible rows may be very large or even infinite, making this approach impractical. To make it more practical, we split the space of possible input rows into "equivalence classes" such that for each row $\mathbf{r}$ in an equivalence class X, the derivative $\mathsf{Count}_{\sigma_{\theta[\mathbf{r}/T_i]}(\prod_{j\neq i} T_j)}(D)$ is the same. Sometimes, we may use an upper bound instead of the actual derivative and require only the upper bounds to be the same within an equivalence class. This reduces the number of equivalence classes. Each equivalence class can contain rows from only one table.

When computing an upper bound of

$$\mathsf{DSC}_{\sigma_\theta(\prod_j T_j)}(D) \le \max_{T,\mathbf{r}} \sum_{i:\ T_i=T} \mathsf{Count}_{\sigma_{\theta[\mathbf{r}/T_i]}(\prod_{j\neq i} T_j)}(D)$$

the quantity $\mathsf{Count}_{\sigma_{\theta[\mathbf{r}/T_i]}(\prod_{j\neq i} T_j)}(D)$ will then depend on both $\mathbf{r}$ and i instead of only i (which would make the number of different variants finite). The number of possible $\mathbf{r}$ can be very large. This is where the equivalence classes become useful. Suppose that $\theta = \theta_1 \wedge \ldots \wedge \theta_n$ and some of the θ_k represent an equality constraint that equates two columns (these are often used for joining two tables). We take the transitive closure $\mathcal{T}$ of this set of equality constraints. When going through the elements $\tilde{\mathbf{r}}$ of $\prod_{j\neq i} T_j$, we use $\mathcal{T}$ to determine which cells of $\mathbf{r}$ are equal to some of the cells of $\tilde{\mathbf{r}}$. These cells can be bound to concrete values in the expression θ. The values in other cells of $\mathbf{r}$ should range over all elements of C; instead of enumerating all their possible values, we existentially quantify these cells.

Such $\mathbf{r}$ is represented as a pattern $\mathbf{p} \in \ell(T) \to \mathsf{C} \cup \{*\}$, where some cells have concrete values but the rest (which are existentially quantified) are represented as $*$. For fixed i, the cells where the $*$ occur are the same for all obtained patterns. Thus the subsets of $\mathsf{C}^{\ell(T_i)}$ represented by different obtained patterns are disjoint and we can just count the number of times each pattern is obtained. If the tables T_i are all different then number of summands in

$$\max_{T,\mathbf{r}} \sum_{i:\ T_i=T} \mathsf{Count}_{\sigma_\theta(\prod_{j\neq i} T_j)}(D)$$

is 1 and we can simply take the maximum of the counts of the patterns.

However, if the number of summands is greater than 1 then the $*$ do not necessarily occur in the same places in all the patterns. Then we need to unify the patterns. For example, if from one summand we obtain a pattern

$$(*, 100, *)$$

with count 5 and from another summand the pattern

$$(1000, *, *)$$

with count 10, then we unify the two patterns and get

$$(1000, 100, *)$$

with count $5 + 10 = 15$. In general, we must unify many such patterns. Then we must consider the unifications of all the subsets of patterns because the unification of a larger subset might not match anything. To find all those unifications, we use a divide-and-conquer approach, splitting the set of

patterns in half, unifying (and sorting) each half recursively, and then merging the results, similarly to merge sort.

After unification, we have a set of pairs $(\mathbf{p}, d)$, each denoting that for all rows $\mathbf{r}$ matched by the pattern $\mathbf{p}$ but not by a more specific pattern in the obtained pattern set, there is an upper bound for the derivative:

$$\frac{\partial}{\partial x_{T;\mathbf{r}}}(\mathsf{Count}_{\sigma_\theta(\prod_j T_j)}(D)) \leq d \ .$$

Also, each row $\mathbf{r}$ for which the derivative is nonzero, is matched by at least one obtained pattern, exactly one of these patterns is the most specific (the one that is the unification of all obtained patterns matching $\mathbf{r}$) and this has the maximum d among the matching patterns.

Now, to compute an upper bound of $\mathsf{DSC}_{\sigma_\theta(\prod_j T_j)}(D)$, we simply have to take the maximum of the obtained d:

$$\mathsf{DSC}_{\sigma_\theta(\prod_j T_j)}(D) \leq \max_{\mathbf{p}} d = \max_{\mathbf{p}} \sum_{i:\ \mathsf{R}(\mathbf{p}) \subseteq \mathsf{C}^{\ell(T_i)}} \mathsf{Count}_{\sigma_{\theta[\mathbf{p}/T_i]}(\prod_{j\neq i} T_j)}(D)$$

Here $\mathbf{p}$ ranges over the set of obtained patterns, $R(\mathbf{p})$ is the set of rows matched by the pattern $\mathbf{p}$. The summation here is the one performed during the unification of patterns, each summand denotes the count of one of the original patterns before unification. $\theta[\mathbf{p}/T_i]$ denotes that the columns of T_i (except those for which there is $*$ in the pattern) in θ there are bound to the values in $\mathbf{p}$ (i.e. replaced by constants).

Let us now describe in more details how pattern unification works.

Unification of patterns. Let us consider a table T used in the query. It may be used more than once, i.e. $T = T_i$ may hold for more than one i. Let $\mathcal{I} = \{i \mid T = T_i\}$. The set of possible rows in T is $\mathcal{R} = \mathsf{C}^{\ell(T)}$. This is the set of rows whose addition or removal is considered when determining the sensitivity of a query (as opposed to the set of rows D^T in the table T in the actual database D). The set of possible patterns is $\mathcal{P} = (\mathsf{C} \cup \{*\})^{\ell(T)} \cup \{\bot\}$. We have added the null pattern $\bot$ that does not match any row. The set of rows matched by a pattern $\mathbf{p} \in \mathcal{P}$, is $\rho(\mathbf{p}) = \{\mathbf{r} \in \mathcal{R} \mid \forall s \in \ell(T).\ \mathbf{p}(s) = \mathbf{r}(s) \vee \mathbf{p}(s) = *\}$ if $p \neq \bot$, and $\rho(\bot) = \emptyset$. The set of rows matched by a pattern set $P \subseteq \mathcal{P}$ is $\bigcup_{\mathbf{p}\in P} \rho(\mathbf{p})$. The *intersection* of two patterns $\mathbf{p}_1, \mathbf{p}_2 \in \mathcal{P}$ is a pattern $\mathbf{p}_1 \cap \mathbf{p}_2 \in \mathcal{P}$ such that $\rho(\mathbf{p}_1) \cap \rho(\mathbf{p}_2) = \rho(\mathbf{p}_1 \cap \mathbf{p}_2)$. It is easy to see that $\mathbf{p}_1 \cap \mathbf{p}_2$ exists and is unique. A set of patterns $P \subseteq \mathcal{P}$ is *disjoint* if $\forall \mathbf{p}_1, \mathbf{p}_2 \in P.\ \mathbf{p}_1 \neq \mathbf{p}_2 \Rightarrow \rho(\mathbf{p}_1) \cap \rho(\mathbf{p}_2) = \emptyset$.

Consider a fixed $i \in \mathcal{I}$. Before, we went through $\sigma_{\theta'_i}(\prod_{j\neq i} T_j)$ and obtained a disjoint set of patterns P_i with a count (multiplicity) for each pattern. Let us call a set of patterns with a (possibly fractional) multiplicity (in $\mathbb{R}_+$) for each pattern, where the multiplicity $M(\mathbf{p})$ of a pattern $\mathbf{p}$ is monotonically decreasing in the set of matched rows of the pattern ($\rho(\mathbf{p}_1) \subseteq \rho(\mathbf{p}_2) \Rightarrow M(\mathbf{p}_1) \geq M(\mathbf{p}_2)$), a *pattern map*. We would like to take the union $\bigcup_{i\in\mathcal{I}} P_i$ but this is not necessarily disjoint. Thus a row may be matched by more than one pattern, making it difficult to find the represented row(s) with the highest multiplicity.

We need a more general class of pattern sets which can represent $\bigcup_{i\in\mathcal{I}} \rho(P_i)$ but each row would be represented by only one pattern in the set. Let P be a pattern set and let a row $\mathbf{r}$ be represented by the most specific pattern $\mathbf{p} \in P$ that matches $\mathbf{r}$, i.e. $\rho(\mathbf{p}) \subseteq \rho(\mathbf{p}')$ for all $\mathbf{p}' \in P$ that match $\mathbf{r}$. Then $\rho(\mathbf{p}) = \bigcap\{\rho(\mathbf{p}') \mid \mathbf{p}' \in P, \mathbf{r} \in \rho(\mathbf{p}')\}$ and $\mathbf{p} = \bigcap\{\mathbf{p}' \mid \mathbf{p}' \in P, \mathbf{r} \in \rho(\mathbf{p}')\}$. Thus $\mathbf{p}$ is unique. To ensure that $\mathbf{p}$ exists, we require that P be closed under intersection. We call such a P a *closed pattern set*. A disjoint pattern set can be converted to a closed pattern set by adding the null pattern.

Suppose now that we have two closed pattern sets $P_1, P_2 \in \mathcal{P}$. How do we find a closed pattern set $P \in \mathcal{P}$ such that $\rho(P) = \rho(P_1) \cup \rho(P_2)$? The naive method is to compute P as $P_1 \cup P_2 \cup \{\mathbf{p}_1 \cap \mathbf{p}_2 \mid \mathbf{p}_1 \in P_1, \mathbf{p}_2 \in P_2\}$. This has complexity $\Omega(|P_1| \cdot |P_2|)$. We will instead use a method similar to the merge of merge sort, having complexity $O(|P_1| + |P_2| + |P|)$ if the length of patterns $|\ell(T)|$ is considered constant (it may be exponential in $\ell(T)$, at most $O(3^{\ell(T)})$, due to the three recursive MergeCPS calls when computing B_1, B_2, B_3; but $\ell(T)$ is usually small in practice and the worst case is probably not reached very easily, unless $B_1 = B_2 = B_3$). $|P|$ is also $\Theta(|P_1| \cdot |P_2|)$ in the worst case but often is much less in practice. This method assumes that P_1 and P_2 are input lexicographically sorted ($\bot$ is considered as the smallest element, and $*$ is larger than any element of C). Then P is also output lexicographically sorted. Let $\top$ be the largest element of $\mathcal{P}$, i.e. $\top = (*, \ldots, *)$. Then $P \cup \{\top\} = \{\mathbf{p}_1 \cap \mathbf{p}_2 \mid \mathbf{p}_1 \in P_1 \cup \{\top\}, \mathbf{p}_2 \in P_2 \cup \{\top\}\}$.

- **function** MergeCPS(P_1, P_2):
  - **Input:** P_1 and P_2 are lexicographically sorted lists (without repetitions) of patterns, which are also closed pattern sets when viewed as sets with added $\bot$. All patterns in P_1 and P_2 must have the same length.
  - **Output:** A lexicographically sorted list (without repetitions) of patterns P, which is also a closed pattern set when viewed as a set with added $\bot$. Also $P = \{\mathbf{p}_1 \cap \mathbf{p}_2 \mid \mathbf{p}_1 \in P_1, \mathbf{p}_2 \in P_2\}$ holds when P, P_1, P_2 are viewed as sets.
  - If P_1 or P_2 (or both) is an empty list then **return** an empty list.
  - If the length of the patterns in the input is zero then **return** a list containing only the empty pattern (the pattern () with length zero).
  - Let A be the sorted list of elements that occur as the first component of some pattern in P_1 or P_2. Always include $*$ in A even if does not occur as the first component of any pattern.
  - For all $i \in \{1, 2\}, a \in A$, let P_{ia} be the lexicographically sorted (possibly empty) list of the patterns in P_i that begin with a, with the first component (a) removed.
  - Let $Q_* = \text{MergeCPS}(P_{1*}, P_{2*})$.
  - For all $a \in A \setminus \{*\}$, let $Q_a = \text{SimpleMerge}(B_1, B_2, B_3)$, where $B_1 = \text{MergeCPS}(P_{1a}, P_{2a})$, $B_2 = \text{MergeCPS}(P_{1*}, P_{2a})$, $B_3 = \text{MergeCPS}(P_{1a}, P_{2*})$.
  - For all $a \in A$, let Q'_a be the list obtained from Q_a by prepending a to each pattern of Q_a.
  - Let P be the concatenation of the lists Q'_a over $a \in A$ in ordered by a.
  - **Return** P.
- **function** SimpleMerge(B_1, B_2, B_3) merges three sorted lists (without repetitions) and removes the repetitions of elements.

Algorithm 3: Merging closed pattern sets

Algorithm 3 computes $\{\mathbf{p}_1 \cap \mathbf{p}_2 \mid \mathbf{p}_1 \in P_1, \mathbf{p}_2 \in P_2\}$, i.e. it merges closed pattern sets. We actually needed to merge pattern maps, i.e. for each pattern $\mathbf{p} = \mathbf{p}_1 \cap \mathbf{p}_2$ in the merged pattern set we also need to compute the multiplicity $M(\mathbf{p}) = M(\mathbf{p}_1) + M(\mathbf{p}_2)$. It is easy to see that the monotonicity of the multiplicity is maintained by the merge (because $\mathbf{p} = \mathbf{p}_1 \cap \mathbf{p}_2 \wedge \mathbf{p}' = \mathbf{p}'_1 \cap \mathbf{p}'_2 \wedge \rho(\mathbf{p}) \subseteq \rho(\mathbf{p}') \Rightarrow \rho(\mathbf{p}_1) \subseteq \rho(\mathbf{p}'_1) \wedge \rho(\mathbf{p}_2) \subseteq \rho(\mathbf{p}'_2)$). The monotonicity ensures that the most specific pattern which represents a row gives the maximum multiplicity. To compute the multiplicities, we need to slightly augment Algorithm 3. In SimpleMerge, when merging two equal patterns, we retain the copy with the maximum multiplicity (i.e. the one obtained from the most specific patterns). In MergeCPS, multiplicities are added when merging patterns with length zero. Elsewhere, multiplicity is always attached to each pattern and it is not changed when the first component of a pattern is removed or added back.

Now we can merge two closed pattern maps. If we need to merge more than two (i.e. $|\mathcal{I}| > 2$), we split the set of pattern maps in half, merge each half recursively and then merge the results, similarly to merge sort. The initial pattern maps need to be lexicographically sorted. If they are not then we can consider each pattern (with multiplicity) in the pattern maps as a separate pattern map and merge the many small pattern maps recursively. This is the approach taken in the implementation.

Smoothing the derivative sensitivity. After computing the derivative sensitivity, we need to find a smooth upper bound of this. We smoothe $\mathsf{Count}_{\sigma_{\theta[\mathbf{p}/T_i]}(\prod_{j \neq i} T_j)}(D)$ separately for each $\mathbf{p}, i$. This gives us a smooth upper bound for the whole expression because the smoothing distributes over maximum and sum. In the following, we write simply θ instead of $\theta[\mathbf{p}/T_i]$. Let

$$A^{(k)}(D) = \max_{d(D,D')=k} \mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}(D') \ . \tag{33}$$

This is similar to the *sensitivity at distance k* from [52]. In our case, k need not be an integer. We can actually generalize $d(D, D')$ even more. Let us assume that the difference in rows for one table can

be more significant than the different of rows in another tables. That is, we can assign a weight G_i to the table T_i, so that

$$d((T_1, T_2, \ldots, T_m), (T'_1, T'_2, \ldots, T'_m)) = \sum_{i=1}^{m} G_i \sum_{x \in X_i} |T_i(x) - T'_i(x)| \ ,$$

$$G_i > 0 \ .$$

Putting $G_i = 1$ for all i, we get a standard definition of sensitivity at distance k. In this section, we will only need $G_i = 1$. The more general result will be needed in Section 3.3.9.

Because $\mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}$ is monotonically increasing, we may consider only those D' that are obtained by adding rows to D. Note that

$$\mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}(D) \leq \mathsf{Count}_{\prod_{j \neq i} \sigma_\theta(T_j)}(D) = \prod_{j \neq i} \mathsf{Count}_{\sigma_\theta(T_j)}(D) \ .$$

Suppose we add k_j rows to T_j for all j, so that $\sum_j G_j k_j = k$. Then

$$\mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}(D') \leq \prod_{j \neq i} \mathsf{Count}_{\sigma_\theta(T_j)}(D') \leq \prod_{j \neq i} (\mathsf{Count}_{\sigma_\theta(T_j)}(D) + k_j) \ .$$

We can get a better bound by noting that every extra row counted in $\mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}(D')$ compared to $\mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}(D)$ is also an extra row in $\mathsf{Count}_{\prod_{j \neq i} \sigma_\theta(T_j)}(D')$ compared to $\mathsf{Count}_{\prod_{j \neq i} \sigma_\theta(T_j)}(D)$. Thus

$$\begin{aligned}
&\mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}(D') - \mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}(D) \leq \mathsf{Count}_{\prod_{j \neq i} \sigma_\theta(T_j)}(D') - \mathsf{Count}_{\prod_{j \neq i} \sigma_\theta(T_j)}(D) = \\
&= \prod_{j \neq i} \mathsf{Count}_{\sigma_\theta(T_j)}(D') - \prod_{j \neq i} \mathsf{Count}_{\sigma_\theta(T_j)}(D) \leq \prod_{j \neq i} (\mathsf{Count}_{\sigma_\theta(T_j)}(D) + k_j) - \prod_{j \neq i} \mathsf{Count}_{\sigma_\theta(T_j)}(D)
\end{aligned}$$

$$\mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}(D') \leq \mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}(D) + \prod_{j \neq i} (\mathsf{Count}_{\sigma_\theta(T_j)}(D) + k_j) - \prod_{j \neq i} \mathsf{Count}_{\sigma_\theta(T_j)}(D) \ .$$

We already described how to compute $\mathsf{Count}_{\sigma_\theta(\prod_{j \neq i} T_j)}(D)$. Now we show how to compute the maximum value of

$$N = \prod_{j \neq i} (\mathsf{Count}_{\sigma_\theta(T_j)}(D) + k_j) - \prod_{j \neq i} \mathsf{Count}_{\sigma_\theta(T_j)}(D) \ .$$

Let $n_j = \mathsf{Count}_{\sigma_\theta(T_j)}(D)$. Then

$$N = \prod_{j \neq i} (n_j + k_j) - \prod_{j \neq i} n_j \ .$$

We want to find $\max_{\sum G_j k_j = k} N$. Note that

$$\frac{\partial N}{\partial k_m} = \prod_{j \notin \{i,m\}} (n_j + k_j) = \frac{1}{n_m + k_m} \prod_{j \neq i} (n_j + k_j)$$

$$\frac{\partial N}{\partial (G_m k_m)} = \frac{1}{G_m (n_m + k_m)} \prod_{j \neq i} (n_j + k_j) \ .$$

The derivative is largest for the m for which $G_m(n_m + k_m)$ is smallest. Let us call the product of G_m and the number of rows in table T_m the modified count of table T_m. Thus we should add rows to the table with the smallest modified count first. After the modified count of this table reaches the modified count of the table with the second smallest modified count, we continue by adding rows to those two tables simultaneously (keeping their modified counts equal), until they reach the modified count of the third smallest table, and so on, until the whole budget of k units of d-distance (where one unit can be used to

change the modified count of a table by 1) has been distributed. This algorithm allows finding an upper bound

$$A(k) = \mathsf{Count}_{\sigma_\theta(\prod_{j\neq i} T_j)}(D) + \prod_{j\neq i}(n_j + k_j) - \prod_{j\neq i} n_j$$

of $A^{(k)}(D)$, but we are interested in finding the smooth upper bound

$$\max_k e^{-k\beta} A(k) \ .$$

Consider first the logarithm of the argument of max:

$$L = -k\beta + \ln A(k)$$

and the derivative of the logarithm

$$\frac{\partial L}{\partial k} = -\beta + \frac{1}{A(k)} \cdot \frac{\partial(A(k))}{\partial k} \ .$$

The derivative of $A(k)$ is

$$\frac{\partial(A(k))}{\partial k} = \frac{\partial N}{\partial(G_m k_m)} = \frac{1}{G_m(n_m + k_m)} \prod_{j\neq i}(n_j + k_j)$$

where $m = \operatorname{argmin}_j(G_j(n_j + k_j))$. Also

$$A(k) = \prod_{j\neq i}(n_j + k_j) - C$$

where $C = \prod_{j\neq i} n_j - \mathsf{Count}_{\sigma_\theta(\prod_{j\neq i} T_j)}(D) \geq 0$. Now

$$\frac{\partial L}{\partial k} = -\beta + \frac{1}{G_m(n_m + k_m)} \cdot \frac{\prod_{j\neq i}(n_j + k_j)}{\prod_{j\neq i}(n_j + k_j) - C} = -\beta + \frac{1}{G_m(n_m + k_m)} \cdot \left(1 + \frac{C}{\prod_{j\neq i}(n_j + k_j) - C}\right)$$

The quantity $\frac{1}{G_m(n_m+k_m)}$ is decreasing in k when $k > 0$. Also, because $C \geq 0$ and when $k > 0$, $C < \prod_{j\neq i}(n_j + k_j)$, the quantity $1 + \frac{C}{\prod_{j\neq i}(n_j+k_j)-C}$ is positive and monotonically decreasing in k when $k > 0$. Thus also $\frac{\partial L}{\partial k}$ is decreasing in k when $k \in (0, \infty)$. Note also that $\lim_{k\to\infty} \frac{\partial L}{\partial k}(k) = -\beta$. Thus L is either decreasing in $(0, \infty)$, or increasing in $(0, s]$ and decreasing in $[s, \infty)$ for some s. Because L is continuous, we get that L is either decreasing in $[0, \infty)$, or increasing in $[0, s]$ and decreasing in $[s, \infty)$ for some s. The maximum that we want to find is obtained with $k = 0$ in the first case, and with $k = s$ in the second case. We know how to compute $\frac{\partial L}{\partial k}(k)$. First try to compute $\frac{\partial L}{\partial k}(0)$. If it is not defined (a division by zero occurs) then $\lim_{k\to 0+} \frac{\partial L}{\partial k}(k) = +\infty$ and we have the second case. Otherwise if $\frac{\partial L}{\partial k}(0) \leq 0$ then we have the first case, and if $\frac{\partial L}{\partial k}(0) > 0$ then we have the second case. In the first case, we have the result. In the second case, we use binary search to find s, which is the value of k where $\frac{\partial L}{\partial k}(k)$ changes from positive to negative.

Let the θ now again denote a general boolean expression, not $\theta[p/T_i]$. Now we have found the $k = s$ that maximizes L and thus also $e^L = e^{-k\beta} A(k)$. This maximum of e^L is a β-smooth upper bound of $\mathsf{Count}_{\sigma_{\theta[p/T_i]}(\prod_{j\neq i} T_j)}(D)$. We compute these upper bounds for each p, i and take the sum and the maximum to get a β-smooth upper bound of $\mathsf{DSC}_{\sigma_\theta(\prod_j T_j)}(D)$.

Now we know how to compute a β-smooth upper bound of the derivative sensitivity of queries of the form $\sigma_\theta(\prod_j T_j)$. We can slightly generalize it to include a projection and renaming of columns at the top level, as these do not change the count or its derivatives. Thus we can compute a β-smooth upper bound of the derivative sensitivity of a query of the form $\rho_{s_1\cdots s_k \to s'_1 \cdots s'_k}(\pi_S(\sigma_\theta(\prod_j T_j)))$ which is the same as that of the query $\sigma_\theta(\prod_j T_j)$.

3.3.8 Derivative Sensitivity for Components. Sensitivity w.r.t. component has been defined in Def. 3.19-3.22 of Sec. 3.3.5. These definitions are given in terms of *global* sensitivity, i.e. sensitivity that does not depend on the actual input. In this section, we extend the notion of sensitivity w.r.t. component to local and derivative. We mostly deal with A-sensitivity, and a bit with B-sensitivity (Def. 3.19-3.20). It is more difficult to give intuitive meaning to differential privacy using C- and D-sensitivity (Def. 3.21-3.22), and we have shown in Theorem 3.12 that they equal to A- and B-sensitivities using Hausdorff distances that we will consider in this section.

3.3.8.1 Differential Privacy and Local Sensitivity w.r.t. Component. We want a definition that guarantees indistinguishability for data instances x and x' that differ in component ρ by a certain amount, but are otherwise as similar as possible. The definitions of A-B-C-D-sensitivities w.r.t. components are aimed to match this definition and provide means of achieving it.

In Sec. 3.3.5, for an element $x \in X$, we called an element $x' \in X'$ that is the closest to x a *friend* of x in the set X' (Definition 3.16), denoted $x \overset{X'}{\sim} x'$. Let us use a shorthand notation $x \overset{\rho}{\sim} x'$ for $x \overset{[x']_\rho}{\sim} x'$. Using $x \overset{\rho}{\sim} x'$ to express the property "x and x' as similar as possible while ρ is fixed", we get the following definitions of differential privacy w.r.t. component.

Definition 3.42 (differential A-privacy w.r.t. component)**.** Let X be a metric space, and $f : X \to D(Y)$. The mapping f is *ε-differentially A-private w.r.t. component ρ* if for all $Y' \subseteq Y$ and for all $x, x' \in X$ such that $x \overset{\rho}{\sim} x'$ and $d_X(x, x') = 1$, the following inequality holds:

$$Pr[f(x) \in Y'] \leq e^{\varepsilon} Pr[f(x') \in Y'] \ .$$

Definition 3.43 (differential B-privacy w.r.t. component)**.** Let X be a metric space, and $f : X \to D(Y)$. The mapping f is *ε-differentially B-private w.r.t. component ρ* if for all $Y' \subseteq Y$ and for all $x, x' \in X$ such that $x \overset{\rho}{\sim} x'$ and $d_X([x]_\rho, [x']_\rho) = 1$, the following inequality holds:

$$Pr[f(x) \in Y'] \leq e^{\varepsilon} Pr[f(x') \in Y'] \ .$$

The definition of (ε, δ)-differential privacy would be analogous. Both Def. 3.42 and Def. 3.43 are suitable for cases where the data owner is interested in concealing changes in particular rows and columns of his datatables. For example, it could be the case where the table is shared by several owners, and one owner knows only some particular rows or columns in the database, and wants to protect his own data independently of the values belonging to the other data owners. While Def. 3.42 allows us to compute more precise bounds for the added noise, Def. 3.43 is better for capturing the change in a particular class ρ, and corresponds to our intuition of hiding a particular component. If we are dealing with components that are vector coordinates, and the distance corresponds to ℓ_p-norm, we actually have $d_X([x]_\rho, [x']_\rho) = d_X(x, x')$.

Local sensitivity w.r.t. component. The difference from the global sensitivity is that the local sensitivity may depend on the input. Similarly to *global* sensitivity w.r.t. component (Definitions 3.19-3.22), we list four slightly different definitions of *local* sensitivity w.r.t. component, now allowing sensitivity to depend on the function input.

Definition 3.44 (local A-B-C-D- sensitivity w.r.t. component)**.** Let $c : \mathcal{P}(X) \to \mathbb{R}^+$, where $\mathcal{P}(X)$ denotes the powerset of X. Mapping $f : X \to Y$ is called locally c-sensitive w.r.t. the component ρ if for all $X_1, X_2 \in X/\rho$ we have

A- $\forall x_1 \in X_1, x_2 \in X_2 : \ x_1 \overset{X_2}{\sim} x_2 \implies d_Y(f(x_1), f(x_2)) \leq c(X_1) \cdot d_X(x_1, x_2)$;

B- $\forall x_1 \in X_1, x_2 \in X_2 : \ x_1 \overset{X_2}{\sim} x_2 \implies d_Y(f(x_1), f(x_2)) \leq c(X_1) \cdot \tilde{d}_X(X_1, X_2)$;

C- $\forall X_1' \subseteq X_1, X_2' \subseteq X_2 : \ X_1' \overset{X_2}{\sim} X_2' \implies \tilde{d}_Y(f(X_1'), f(X_2')) \leq c(X_1) \cdot \tilde{d}_X(X_1', X_2')$;

D- $\forall X_1' \subseteq X_1, X_2' \subseteq X_2 : \ X_1' \overset{X_2}{\sim} X_2' \implies \tilde{d}_Y(f(X_1'), f(X_2')) \leq c(X_1) \cdot \tilde{d}_X(X_1, X_2)$.

Alternatively, we could define $c : X \to \mathbb{R}^+$ on particular points $x_1 \in X_1$. Such definition would give more precise bounds since $c(x)$ needs to give a suitable bound only for x, and not necessarily for all other elements of $[x]_\rho$. On the other hand, $c([x]_\rho)$ is more useful if some data is missing and only the ρ-class of the input is known, e.g. in the case of shared database.

The proofs of local sensitivities w.r.t. composed components are analogous to those of global sensitivity. As an example, we show the composition proof local A-sensitivity, and the other types of local sensitivities are analogous. We see that another reason for taking $c([x]_\rho)$ is composability.

Theorem 3.57. *Let (ρ, σ) be an adjacent component pair (Def. 3.23). Let the mapping $f : X \to Y$ be c_ρ and c_σ locally A-sensitive w.r.t. ρ and σ respectively. Then, f is $(c_\rho + c_\sigma)$ locally A-sensitive w.r.t $\rho \sqcap \sigma$.*

Proof. Let $x, x' \in X$ be such that $x \overset{[x']_{\rho\sqcap\sigma}}{\sim} x'$. We need to show that $d_Y(f(x), f(x')) \leq (c_\rho + c_\sigma)([x]_{\rho\sqcap\sigma}) \cdot d_X(x, x')$. Since (ρ, σ) is a adjacent pair, there exists $x'' \in X$ such that $x \overset{[x']_\rho}{\sim} x''$ and $x' \overset{[x]_\sigma}{\sim} x''$. Since $x'' \in [x']_\rho$ and $x'' \in [x]_\sigma$, we have $[x'']_\rho = [x']_\rho$, $[x'']_\sigma = [x]_\sigma$, and $x \overset{[x'']_\rho}{\sim} x''$, $x' \overset{[x'']_\sigma}{\sim} x''$, so we can apply A-sensitivity.

- $d_Y(f(x), f(x'')) \leq c_\rho([\hat{x}]_\rho) \cdot d_X(x, x'')$ for $\hat{x} \in \{x, x''\}$;
- $d_Y(f(x''), f(x')) \leq c_\sigma([\bar{x}]_\sigma) \cdot d_X(x'', x')$ for $\bar{x} \in \{x', x''\}$.

Since $x \overset{[x']_\rho}{\sim} x''$, the element x' cannot be closer to x than x'' is, so $d_X(x, x'') \leq d_X(x, x')$.

Since $x' \overset{[x]_\sigma}{\sim} x''$, the element x cannot be closer to x' than x'' is, so $d_X(x'', x') \leq d_X(x, x')$.

We now estimate $d_Y(f(x), f(x'))$ from above, using the triangle inequality. The tricky part is that the definition of local sensitivity does not allow to use $c_\sigma([x]_\sigma)$ as a multiplier for $d_X(x'', x')$, and $c_\rho([x']_\rho)$ as a multiplier for $d_X(x'', x)$. However, we would like to get new sensitivity $c_{\rho\sigma}$ that would depend either only on x, or only on x'. Since the intermediate point x'' shares its σ-class with x, we have $c_\sigma([x'']_\sigma) = c_\sigma([x]_\sigma)$. Taking $\hat{x} := x$ and $\bar{x} := x''$ gives us

$$\begin{aligned} d_Y(f(x), f(x')) &\leq d_Y(f(x), f(x'')) + d_Y(f(x''), f(x')) \\ &\leq c_\rho([x]_\rho) \cdot d_X(x, x'') + c_\sigma([x'']_\sigma) \cdot d_X(x'', x') \\ &\leq c_\rho([x]_\rho) \cdot d_X(x, x') + c_\sigma([x'']_\sigma) \cdot d_X(x, x') \\ &= c_\rho([x]_\rho) \cdot d_X(x, x') + c_\sigma([x]_\sigma) \cdot d_X(x, x') \\ &= (c_\rho + c_\sigma)([x]_{\rho\sqcap\sigma}) \cdot d_X(x, x') \ . \end{aligned}$$

Here the function addition is defined as $(c_\rho + c_\sigma)([x]_{\rho\sqcap\sigma}) := c_\rho([x]_\rho) + c_\sigma([x]_\sigma)$. The definition is correct, since the classes $[x]_\rho$ and $[x]_\sigma$ are uniquely determined by $[x]_{\rho\sqcap\sigma}$. □

Sensitivity w.r.t. basis change. Let us now assume that the distances in X are ℓ_p-norms, and their generalization to sets is a Hamming distance – we want that the vector distances would be subadditive w.r.t. distances of their coordinates.

Let $X = (X_1, \ldots, X_n)$ be a set of vectors $\vec{x} = (x_1, \ldots, x_n)$, where the coordinates x_i are independent. Suppose that we know the sensitivity c_i of some function $f : X \to Y$ w.r.t. each component ρ_i, which is the i-th coordinate of $\vec{x}$. We want to find the sensitivities of the same function w.r.t. some *other* set of components $\vec{z} = (z_1, \ldots, z_m)$, such that x_i is uniquely determined by $z_1, \ldots, z_m$ for all $i \in [n]$. We may think that the vector $\vec{z}$ represents m equivalence classes of $\vec{x}$ w.r.t. m different components, such that the m classes uniquely determine $\vec{x}$. Let $g : Z \to X$ be such that $\vec{x} = g(\vec{z})$. We are interested in sensitivity of the function $f \circ g$ w.r.t. the component σ_i that corresponds to the i-th coordinate of $\vec{z}$.

First of all, we may take Theorem 3.31 and extend it to *local* sensitivity.

Theorem 3.58. *Let $g : Z \to X$ be locally c_g sensitive w.r.t. component σ. Let $f : X \to Y$ be locally c_f sensitive. Then, the function composition $f \circ g$ is $\lambda Z'.c_g(Z') \cdot c_f(\sup\{c_f(g(z)) | z \in Z'\})$ sensitive w.r.t. σ, using any definition of sensitivity (A-,B-,C-,D-).*

Proof. If global sensitivity was considered as in Theorem 3.31, we would get global sensitivity $c_g \cdot c_f$ in all four cases. For local sensitivity, this value would also depend on the particular input. First, assume that only c_g is local, and c_f is global. For $z \in Z$, we would directly get $c_g([z]_\sigma) \cdot c_f$ (A- and B- sensitivity). We would also get $c_g(Z') \cdot c_f$ for a subset of a class $Z' \in Z/\sigma$ (C- and D- sensitivity).

We may actually obtain a better bound than the global sensitivity c_f, since $g(z)$ does not take arbitrary values, and the set of possible inputs for f is constrained. Knowing that $z \in Z'$, the set of possible inputs for c_f is $\{g(z)|z \in Z'\}$. We take sup of c_f applied to this set. □

A similar bound can be obtained if we treat g as a set functions $g_1, \ldots, g_m$, each g_i mapping z to a particular equivalence class of x. This is stated in the next theorem.

Theorem 3.59. *Let $\rho_1, \ldots, \rho_n$ be pairwise adjacent equidistant components of a set X, such that $X/\rho_1 \times \cdots \times X/\rho_n \sim X$, and $h : X/\rho_1 \times \cdots \times X/\rho_n \to X$ is the corresponding natural isomorphism. Let $f : X \to Y$, and let c_i be the local sensitivity of f w.r.t. ρ_i for all $i \in [n]$. Let Z be a set, and let $g_i : Z \to X/\rho_i$ be functions that satisfy $\forall x \in X\ \exists z \in Z\ \forall i \in [n] : [x]_{\rho_i} = g_i(z)$. Let $\sigma_1, \ldots, \sigma_m$ be pairwise adjacent equidistant components of Z, and c^i_j the sensitivity of g_i w.r.t. σ_j. Let $g : Z \to X$ be defined as $g(z) := h(g_1(z), \ldots, g_n(z))$. The sensitivity of the function $f \circ g$ w.r.t. the component σ_j is $c(Z') = \sum_{i=1}^n \sup\{c_i(g_i(z))|z \in Z'\} \cdot c^i_j(Z')$.*

Proof. Let local sensitivity w.r.t. component be defined as in Def. 3.44. Since the components ρ_i and σ_i are equidistant, A- and B-sensitivity are equivalent, so without loss of generality, let us consider A-sensitivity. We have

$$x \overset{[x']_{\rho_i}}{\sim} x' \implies d_Y(f(x), f(x')) \leq c_i([x]_{\rho_i}) \cdot d_X(x, x') \ .$$

Since the components ρ_i are pairwise adjacent, for any pair $x, x' \in X$, there is a sequence of $n+1$ elements x^i starting with $x^1 = x$ and ending with $x^{n+1} = x'$, where $[x^i]_{\rho_i} = [x]_{\rho_i}$, $[x^{i+1}]_{\rho_i} = [x']_{\rho_i}$, and $x^i \overset{[x']_{\rho_i}}{\sim} x^{i+1}$ for all $i \in [n]$. Using triangle inequality, for all $x, x' \in X$, we get

$$d_Y(f(x), f(x')) \leq \sum_{i=1}^n d_Y(f(x^i), f(x^{i+1})) \leq \sum_{i=1}^n c_i([x^i]_{\rho_i}) \cdot d_X(x^i, x^{i+1}) \ .$$

Since the components ρ_i are equidistant, and $x^i \overset{[x']_{\rho_i}}{\sim} x^{i+1}$, we have $d_X(x^i, x^{i+1}) = \tilde{d}_X([x]_{\rho_i}, [x']_{\rho_i})$ for all $i \in [n]$. On the other hand, there exist $z, z' \in Z$ such that $g_i(z) = [x]_{\rho_i}$ and $g_i(z') = [x']_{\rho_i}$ for all $i \in [n]$. Together, this gives us

$$\begin{aligned} d_Y(f(x), f(x')) &\leq \sum_{i=1}^n c_i([x]_{\rho_i}) \cdot d_X(x^i, x^{i+1}) \\ &= \sum_{i=1}^n c_i([x]_{\rho_i}) \cdot \tilde{d}_X([x]_{\rho_i}, [x']_{\rho_i}) \\ &= \sum_{i=1}^n c_i([x]_{\rho_i}) \cdot \tilde{d}_X(g_i(z), g_i(z')) \ . \end{aligned}$$

To proceed further, we need to know the sensitivities of functions g_i w.r.t. distances $\tilde{d}_X(g_i(z), g_i(z'))$. Since we estimate the local sensitivity of $f \circ g$ w.r.t. the component σ_j, we assume $z \overset{[z']_{\sigma_j}}{\sim} z'$. Let c^i_j be the local sensitivity of g_i w.r.t. σ_j. By definition of local sensitivity w.r.t. component, we have

$\tilde{d}_X(g_i(z), g_i(z')) \leq c_j^i([z]_{\sigma_j}) \cdot d_Z(z, z')$. We get

$$\begin{aligned} d_Y(f(x), f(x')) &\leq \sum_{i=1}^{n} c_i([x]_{\rho_i}) \cdot \tilde{d}_X(g_i(z), g_i(z')) \\ &\leq \sum_{i=1}^{n} c_i([x]_{\rho_i}) \cdot c_j^i([z]_{\sigma_j}) \cdot d_Z(z, z') \\ &\leq \sum_{i=1}^{n} c_i(g_i(z)) \cdot c_j^i([z]_{\sigma_j}) \cdot d_Z(z, z') \ . \end{aligned}$$

By definition, $g(z) = h(g_1(z), \ldots, g_n(z)) = h([x]_{\rho_1}, \ldots, [x]_{\rho_n})$, and since h is the natural isomorphism, we have $h([x]_{\rho_1}, \ldots, [x]_{\rho_n}) = x$. Hence, we may substitute $x = g(z)$, $x' = g(z')$, and get

$$d_Y(fg(z), fg(z')) \leq \sum_{i=1}^{n} c_i(g_i(z)) \cdot c_j^i([z]_{\sigma_j}) \cdot d_Z(z, z') \ .$$

This still does not look like local sensitivity w.r.t. component, since $c_i(g_i(z))$ depends on z, not $[z]_{\sigma_j}$. The bound is nevertheless valid and can be used to compute noise for differential privacy, but it is not composable. To make it composable, we may define a valid upper bound $\hat{c}_i(Z') = \sup\{c_i(g_i(z)) \mid z \in Z'\}$ for $Z' \in Z/\sigma_j$, that now depends only on the σ_j-class of z. The sensitivity of $f \circ g$ w.r.t. the component σ_j will then be the function $c(Z') = \sum_{i=1}^{n} \hat{c}_i(Z') \cdot c_j^i(Z')$. □

The same derivation could be applied to global sensitivity. In that case, the sensitivity would be a constant, and we would get

$$d_Y(fg(z), fg(z')) \leq \left(\sum_{i=1}^{n} c_i \cdot c_j^i \right) \cdot d_Z(z, z') \ .$$

Example 3.5. The function g may express transition to polar coordinates: $(x_1, x_2) := (\sqrt{z_1^2 + z_2^2}, \arctan \frac{x_1}{x_2})$. We have $c_1^1 = c_2^1 = 1$, and the functions c_1^2, c_1^2 are more complicated. If we apply to the radius some function f whose sensitivity c_f we already know, we may compute how the output actually depends on the particular coordinates. For If we have $f(x_1, x_2) = 2 \cdot x_1$, then $c_1 = 2$ and $c_2 = 0$. In this case, we do not need to estimate c_1^2 and c_1^2 since they are not used by f. We get that the sensitivity w.r.t. coordinate z_j is $c_1 \cdot c_j^1 = 2$. This means that, if we shift the value of z_1 or z_2 by k, then the final output will differ in at most $2k$. Indeed, we cannot achieve better results for z_1 in this particular case. If $z_2 = 0$, then we have $y = z_1$, so z_1 affects the output as well as y.

Derivative sensitivity w.r.t. component. If we define a multivariate function $f : X_1 \times \cdots \times X_n \to \mathbb{R}$ for some $X_1, \ldots, X_n$, intuitively, we would like the partial derivative $\frac{df_{\vec{x}}}{dX_i}$ (assuming that it is defined in X_i and exists on the point $\vec{x}$) to be related to the derivative sensitivity w.r.t. component X_i, since this value describes how the function output changes with X_i. This is motivated by Definition 3.39 of derivative sensitivity for row multiplicities, where $X_i = \mathbb{R}$ for all i. Like Definition 3.39, it depends on $\vec{x} \in (X_1, \ldots, X_n)$, and, hence, on the other components X_j for $j \neq i$.

In some cases, when only the equivalence class of $\vec{x}$ is known, we may want that the other components should remain "invisible". This can be necessary e.g. in the case where the party who chooses a suitable ϵ for differential privacy sees only some part of the database. An upper bound that depends only on $x_i \in X_i$ would be $\sup_{\vec{z} \in (X_1, \ldots, X_n) : (z_i = x_i)} \| \frac{df_{\vec{z}}}{dX_i} \|$, defined as ∞ if the supremum does not exist. We see that we would get a more precise bound having an entire vector $\vec{x}$, and in practice it may be more reasonable to compute $\frac{df_{\vec{x}}}{dX_i}$ for an actual database $\vec{x}$ using privacy-preserving computation methods.

Complementary components. Let us now consider univariate $f : X \to \mathbb{R}$ (the space X can be multi-dimensional as well). We want to define sensitivity w.r.t. some abstract component ρ of X. If we want to apply the notion of derivative w.r.t. $[x]_\rho$, we first need to rewrite the function f in such a way that, instead of one variable $x \in X$, it takes at least two variables, one of which is $[x]_\rho$, and the other one contains all information that is sufficient to reconstruct x and evaluate f on it. Since we only have ρ, we need to define the other component (let us denote in $\overline{\rho}$) ourselves. It should always be possible to reconstruct x from $[x]_\rho$ and $[x]_{\overline{\rho}}$, and on the other hand, both $[x]_\rho$ and $[x]_{\overline{\rho}}$ should be necessary to reconstruct x, so that sensitivity w.r.t. ρ would not be 0. Taking into account these two properties, we get the following definition.

Definition 3.45 (complementary component)**.** Let ρ be a component of X. The *complementary component* of ρ, denoted $\overline{\rho}$, is the component of X that satisfies $X/(\rho \sqcap \overline{\rho}) \sim X$ and $X/(\rho \sqcup \overline{\rho}) \sim \{X\}$.

In this definition, $\rho \sqcap \overline{\rho}$ is the finest possible component (each element in its own class), and $\rho \sqcup \overline{\rho}$ is the coarsest possible (all elements are in the same class). The component $\overline{\rho}$ that satisfies this definition is in general not unique.

Example 3.6. Let $X = \{a, b, c\}$, and $(a, b) \in \rho$, but $(b, c), (a, c) \notin \rho$. The two possible candidates for $\overline{\rho}$ are:

- σ_1 such that $(a, c) \in \sigma_1$, $(a, b), (b, c) \notin \sigma_1$;
- σ_2 such that $(b, c) \in \sigma_2$, $(a, b), (a, c) \notin \sigma_2$.

The complementary component $\overline{\rho}$ is not unique even for the simple case of vectors with ℓ_p norms. If we have $\vec{x} = (x_1, x_2)$, and ρ is such that $[\vec{x}]_\rho = x_1$, then we may choose $[\vec{x}]_{\overline{\rho}} = x_2$ as well as $[\vec{x}]_{\overline{\rho}} = x_1 + x_2$. In the second case, any function will first need to compute $x_2 = (x_1 + x_2) - x_1$. The choice of $\overline{\rho}$ does not affect the actual sensitivity w.r.t. ρ in any way, but formally, the derivatives are different. The problem is that $[\vec{x}]_{\overline{\rho}} = x_1 + x_2$ changes with x_1, while for $[\vec{x}]_{\overline{\rho}} = x_2$ it remains the same.

We see that the definition of sensitivity w.r.t. ρ is ambiguous, dependent on the particular choice of $\overline{\rho}$. The choice of $\overline{\rho}$ should depend on the distance that defines the relation $x \overset{\rho}{\sim} x'$, describing what we actually want to hide. In Example 3.6, $x \overset{\rho}{\sim} x'$ holds for $d_X((x_1, x_2), (x_1', x_2)) = \left|x_1 - x_1'\right|$, and in general not for $d_X((x_1, x_1 + x_2), (x_1', x_1' + x_2)) = 2\left|x_1 - x_1'\right|$. We want that the definition of $\overline{\rho}$ would be somehow related to the distance $d_X(\cdot, \cdot)$. We may remove the property $X/(\rho \sqcup \overline{\rho}) \sim \{X\}$ if we replace it with an alternative property that prevents $X/\overline{\rho} \subseteq X/\rho$.

Definition 3.46 (orthogonal complement)**.** Let ρ be a component of X. The *orthogonal complement* of ρ, denoted $\overline{\rho}$, is a component of X that satisfies the following properties:

- $X/(\rho \sqcap \overline{\rho}) \sim X$;
- $x \overset{\rho}{\sim} x' \implies [x]_{\overline{\rho}} = [x']_{\overline{\rho}}$.

The orthogonal complement does not exist in general. It does not exist for the components on the set $X = \{a, b, c\}$ considered in Example 3.6. Moreover, the orthogonal complement is still not unique. However, if there are several orthogonal complements, any of them would give a valid bound for x and x' that satisfy $x \overset{\rho}{\sim} x'$.

The finest orthogonal complement can be constructed from ρ as follows. First, define $x \,\overline{\rho}\, x'$ and $x' \,\overline{\rho}\, x$ for all $x \overset{\rho}{\sim} x'$, and then extend the relation to transitive closure. The component $\overline{\rho}$ definitely cannot be finer, since all relations $x \,\overline{\rho}\, x'$ are necessary for a component to be an orthogonal complement, so if $X/(\rho \sqcap \overline{\rho}) \sim X$ does not hold, then the orthogonal complement does not exist.

Definition 3.47. Let ρ be a component of X. The *minimal orthogonal complement* of ρ, denoted $\overline{\rho}$, is the component of X that is defined as follows:

- $x \,\overline{\rho}\, x$ for $x \in X$;
- $x \,\overline{\rho}\, x'$, $x' \,\overline{\rho}\, x$ for all $x \overset{\rho}{\sim} x'$;

- $x \,\overline{\rho}\, x' \wedge x' \,\overline{\rho}\, x'' \implies x' \,\overline{\rho}\, x''$ for all $x, x', x'' \in X$.

The definition is correct since $\overline{\rho}$ is by construction an equivalence relation. We need to show that it is an orthogonal complement.

Conjecture 3.60. The minimal orthogonal complement $\overline{\rho}$ of ρ is an orthogonal complement of ρ whenever $X/(\rho \sqcap \overline{\rho}) \sim X$.

Proof. The condition $X/(\rho \sqcap \overline{\rho}) \sim X$ is an assumption. Since $x \,\overline{\rho}\, x'$ and $x' \,\overline{\rho}\, x$ are defined for all $x \overset{\rho}{\sim} x'$, we have $x \overset{\rho}{\sim} x' \implies [x]_{\overline{\rho}} = [x']_{\overline{\rho}}$. □

We use the notion of minimal orthogonal complement to define derivative sensitivity w.r.t. component.

Definition 3.48 (A-derivative sensitivity w.r.t. component)**.** Let $f : X \to \mathbb{R}$. Let ρ be a component of X. Let $\sigma := \overline{\rho}$ be the minimal orthogonal complement of ρ, and ψ the isomorphism $X/(\rho \sqcap \sigma) \to X$. The *derivative sensitivity w.r.t. component* ρ is the following mapping from X to $\mathbb{R}^+$:

$$\mathrm{DS}_f^{\rho}(x) = \| \frac{d(f \circ \psi)_{([x]_\rho, [x]_\sigma)}}{d\rho} \| \ .$$

Definition 3.49 (B-derivative sensitivity w.r.t. component)**.** Let $f : X \to \mathbb{R}$. Let ρ be a component of X. Let $\sigma := \overline{\rho}$ be the minimal orthogonal complement of ρ, and ψ the isomorphism $X/(\rho \sqcap \sigma) \to X$. The *derivative sensitivity w.r.t. component* ρ is the following mapping from X/ρ to $\mathbb{R}^+$:

$$\hat{\mathrm{DS}}_f^{\rho}(X^{\rho}) = \sup_{x \in X^{\rho}} \| \frac{d(f \circ \psi)_{([x]_\rho, [x]_\sigma)}}{d\rho} \| \ .$$

Intuitively, A- and B-derivative sensitivity w.r.t. component ρ can be used to make a function f A- and B-differentially private w.r.t. ρ, using e.g. the same construction as in Theorem 3.53. For this, we need to define more formally what the derivatives w.r.t. $[x]_\rho$ are. We can use Theorem 3.53 directly only if the set X/ρ is isomorphic to $\mathbb{R}$. Alternatively, we can generalize Theorem 3.53. We will do it in Sec. 3.3.8.2 where we provide more concrete definitions of components and derivative sensitivity w.r.t. them.

Complementary components that depend on the inputs. As discussed in Example 3.6, the minimal orthogonal complement of ρ on $X = \{a, b, c,\}$, where $(a, b) \in \rho$, $(b, c) \notin \rho$, $(a, c) \notin \rho$ does not exist. The problem is that the two candidate components σ_1 and σ_2 are only partially suitable. For $a \overset{\rho}{\sim} c$, we have $[a]_{\sigma_1} = [c]_{\sigma_1}$, and for $b \overset{\rho}{\sim} c$, we have $[b]_{\sigma_2} = [c]_{\sigma_2}$, but neither σ_1 nor σ_2 satisfies both. In practice, we could find separately the sensitivity for $a \overset{\rho}{\sim} c$ and $b \overset{\rho}{\sim} c$ using different definitions of $\overline{\rho}$, and then find an upper bound that satisfies both.

The construction of such set of components is always possible. For all $x, x' \in X$ satisfying $x \overset{\rho}{\sim} x'$, define a unique component $\overline{\rho}_{x,x'}$ such that $y \,\overline{\rho}_{x,x'}\, y' \iff (x = y \wedge x' = y') \vee (x = y' \wedge x' = y)$. Each $\overline{\rho}_{x,x'}$ puts together only elements that have been in different components of ρ, so we have $\forall i \in X \times X :$ $X/(\rho \sqcap \overline{\rho}_i) \sim X$ and $x \overset{\rho}{\sim} x' \implies !\exists\, i \in X \times X :\ [x]_{\overline{\rho}_i} = [x']_{\overline{\rho}_i}$.

Definition 3.50. Let ρ be a component of X. The *minimal orthogonal complement set* of ρ, denoted $(\overline{\rho}_i)_{i \in X \times X}$, is a set of components of X defined as follows:

- $x \,\overline{\rho_i}\, x$ for $x \in X$, $i \in X \times X$;
- $x \,\overline{\rho}_{x,x'}\, x'$ and $x' \,\overline{\rho}_{x,x'}\, x$ for all $x \overset{\rho}{\sim} x'$;
- $x \,\overline{\rho}_i\, x' \wedge x' \,\overline{\rho}_i\, x'' \implies x' \,\overline{\rho}_i\, x''$ for all $x, x', x'' \in X$, $i \in X \times X$.

We can now give a generalization of Definition 3.49.

Definition 3.51 (B-derivative sensitivity w.r.t. component (generalization))**.** Let $f : X \to \mathbb{R}$. Let ρ be a component of X. Let $(\sigma_i)_{i \in X \times X}$ be the minimal orthogonal complement set of ρ, and ψ_i for $i \in X \times X$ the isomorphism $X/(\rho \sqcap \sigma_i) \to X$. The *derivative sensitivity w.r.t. component* ρ is the following mapping from X/ρ to $\mathbb{R}^+$:

$$\tilde{\mathrm{DS}}_f^{\rho}(X^{\rho}) = \sup_{i \in X \times X, x \in X^{\rho}} \| \frac{d(f \circ \psi_i)_{([x]_{\rho},[x]_{\sigma})}}{d\rho} \| \ .$$

While Definition 3.51 looks complicated, the idea is still the same as for generalization of A-sensitivity to B-sensitivity. Namely, given $[x]_{\rho}$, we compute A-sensitivity for all possible $z \in X$ such that $[z]_{\rho} = [x]_{\rho}$ (each z may have its own complementary component σ_z), and take the worst-case (i.e. largest) derivative sensitivity over $z \in X$.

3.3.8.2 Derivative Sensitivity for Banach Spaces. If we want to use derivative sensitivity w.r.t. components in practice, it is not sufficient just to have the definitions. We also need a way to efficiently compute the derivative sensitivity, and convert it to a differential privacy mechanism. In this section, we show how it can be done for *Banach spaces*.

Banach spaces. First, we recall some basics of Banach space theory. Banach spaces do not allow arbitrary metrics and instead require norms but many useful metrics can also be viewed as norms. We will further denote vectors by $\vec{x}$, and norms by $\|\cdot\|_N$, where N specifies the particular norm. Formal definition of a norms and a seminorm is given in Definition 3.14.

Definition 3.52 (Banach space)**.** A *Banach space* is a vector space with a norm that is *complete* (i.e. each converging sequence has a limit).

Banach spaces combine vector spaces with distances, which are necessary for defining differential privacy. The completeness property allows us to define derivatives. Using the norm of a Banach space, we may generalize the notion of continuous function from real numbers to Banach spaces.

Definition 3.53 (Continuous function in Banach space)**.** Let V and W be Banach spaces, and $U \subset V$ an open subset of V. A function $f : U \to W$ is called *continuous* if

$$\forall \epsilon > 0 : \exists \delta > 0 : \|x - x'\|_V \leq \delta \Rightarrow \|f(x) - f(x')\|_W \leq \epsilon.$$

The notion of the derivative of a function can be also extended to Banach spaces.

Definition 3.54 (Fréchet derivative)**.** Let V and W be Banach spaces, and $U \subset V$ an open subset of V. A function $f : U \to W$ is called *Fréchet differentiable at* $x \in U$ if there exists a bounded linear operator $df_x : V \to W$ such that $\lim_{h \to 0} \frac{\|f(x+h) - f(x) - df_x(h)\|_W}{\|h\|_V} = 0$. Such operator df_x is called Fréchet derivative of f at the point x.

The mean value theorem can be generalized to Banach spaces (to a certain extent).

Theorem 3.61 (Mean value theorem ([55], Chapter XII))**.** *Let V and W be Banach spaces, and $U \subset V$ an open subset of V. Let $f : U \to W$, and let $x, x' \in U$. Assume that f is defined and is continuous at each point $(1-t)x + tx'$ for $0 \leq t \leq 1$, and differentiable for $0 < t < 1$. Then there exists $t^* \in (0,1)$ such that*

$$\|f(x) - f(x')\|_W \leq \|df_z\|_{V \to W} \|x - x'\|_V$$

for $z = (1-t^*)x + t^*x'$, where $\|\cdot\|_{V \to W}$ denotes the norm of operator that maps from V to W.

Derivative sensitivity in Banach spaces. Recall that, for row multiplicities, we defined derivative sensitivity as $\mathrm{DS}_f(\vec{x}) = \max_i \left|\frac{\partial f}{\partial x_i}(\vec{x})\right|$, where x_i denotes the i-th component of the vector of variables $\vec{x}$ (Definition 3.39). We extend it to the case where X is any Banach space:

Definition 3.55. Let X be (an open convex subset of) a Banach space. Let $f : X \to \mathbb{R}$. Let f be Fréchet differentiable at each point of X. The *derivative sensitivity* of f is the following mapping from X to $\mathbb{R}_+$, where $\mathbb{R}_+$ denotes the set of all non-negative real numbers:

$$\mathrm{DS}_f(\vec{x}) = \|df_{\vec{x}}\| \ .$$

where $df_{\vec{x}}$ is the Fréchet derivative of f at $\vec{x}$ and $\|df_{\vec{x}}\|$ is the operator norm of $df_{\vec{x}}$.

We generalize the results of Sec. 3.3.7 from $\mathbb{R}$ to Banach spaces. First, we extend the definition of smoothness to the case where X is any Banach space:

Definition 3.56. Let $p : X \to \mathbb{R}$ and $\beta \in \mathbb{R}$. The mapping p is *β-smooth*, if $p(\vec{x}) \le e^{\beta \cdot \|\vec{x}' - \vec{x}\|} \cdot p(\vec{x}')$ for all $\vec{x}, \vec{x}' \in X$.

We provide generalizations of Theorem 3.53 and Theorem 3.56 to the case where X is a Banach space:

Theorem 3.62. *Let $\gamma, b, \beta \in \mathbb{R}_+$, $\gamma > 1$. Let $\epsilon = (\gamma + 1)(b + \beta)$. Let η be a random variable distributed according to GenCauchy(γ). Let c be a β-smooth upper bound on DS_f for a function $f : X \to \mathbb{R}$. Then $g(\vec{x}) : f(\vec{x}) + \frac{c(\vec{x})}{b} \cdot \eta$ is ϵ-differentially private.*

Theorem 3.63. *Let $b, \beta, \epsilon \in \mathbb{R}_+$, $b > 0$, $b + \beta \le \epsilon$. Define $k = 1 + (\epsilon - b)/\beta$. Let $\delta = e^{-k}$. Let η be a random variable distributed according to Lap(1). Let c be a β-smooth upper bound on DS_f for a function $f : X \to \mathbb{R}$, where X is Banach space and d_X is the distance corresponding to the norm of X. Define $g(\vec{x}) := f(\vec{x}) + \frac{c(\vec{x})}{b} \cdot \eta$. Then*

- *for any $\vec{x}_1, \vec{x}_2 \in X$, $(\epsilon \cdot L, 2\delta) \in d_{\mathrm{DP}}(g(\vec{x}_1), g(\vec{x}_2))$, where $L = d_X(\vec{x}_1, \vec{x}_2)$;*
- *(in particular,) g is $(\epsilon, 2e^{\epsilon}\delta)$-differentially private.*

If, additionally for any two points $\vec{x}_1, \vec{x}_2 \in X$ there exists a shortest path h in X, such that c is monotonic along that path, then the factor "2" in previous statements can be removed.

The proofs of these theorems is analogous to the proofs of Theorem 3.53 and Theorem 3.56, so we do not repeat them here. The only difference is that we are using Definition 3.56 for smoothness, Definition 3.54 for differentiability, and the mean value theorem for Banach spaces and Fréchet derivative (Theorem 3.61).

The derivative sensitivity of Definition 3.55 depends on the actual input $\vec{x}$ and hence corresponds to A-sensitivity w.r.t. component (Definition 3.48). It can be generalized to B-sensitivity (Definition 3.49) in a straightforward way by taking $\sup_{z \in X, [\vec{z}]_\rho = [\vec{x}]_\rho}(\mathrm{DS}_f(\vec{z}))$. The partitioning ρ defines the norm $\|\cdot\|$ of the underlying Banach space. While we cannot construct a Banach space for *any* possible ρ, we can do it for certain classes of ρ.

From components to Banach spaces. Formally, a component is an equivalence relation ρ on a set X. This relation can be viewed as a partitioning of X to disjoint subsets. For these subsets, we can define a distance induced by the distance $d_X(\cdot, \cdot)$ on X. In our practical applications, the Hausdorff distance seems to be the most reasonable choice, since it gives an upper bound on the distance between the closest elements of two subsets, and it the context of databases it would mean that we are considering some change in one particular column, letting all other columns be as similar as possible (ideally, they should remain the same, but it is not always possible).

Let $\rho_1, \ldots, \rho_n$ be such that $X/(\rho_1 \sqcap \cdots \sqcap \rho_n) \sim X$. In order to apply derivative sensitivity, we need that the partitionings X/ρ_i would form some Banach spaces, not necessarily over real numbers. The norm in this Banach space should correspond to the distance $\tilde{d}_X(\cdot, \cdot)$ defined on elements of X/ρ_i.

One possibility is to embed a metric space to a Banach space using *Kuratowski embedding*. For any metric space (M, d), denoting by $C_b(M)$ the Banach space of all bounded continuous real-valued functions on M with the supremum norm, we can take the map $\Phi : M \to C_b(M)$ defined by $\Phi(x)(y) := d(x, y) - d(x_0, y)$ for some fixed $x_0 \in M$. This map transforms elements of M to elements of the Banach space $C_b(M)$, and it is an isometry. In our case $M := X/\rho$, and the distance is $\tilde{d}_X(\cdot, \cdot)$. The functions $f : X/\rho \to \mathcal{P}(\mathbb{R})$ should now be transformed to analogous functions $\hat{f} : C_b(X/\rho) \to \mathcal{P}(\mathbb{R})$ as $\hat{f}(X) := f(\Phi^{-1}(X))$. We may take a more elaborated isometry Φ if it is possible.

Example 3.7. Let X be a set of vectors of length n, equipped with ℓ_p-norm. Let ρ_i be defined in a way $x \; \rho_i \; x' \iff x_i = x'_i$. For all $X_1, X_2 \in X/\rho_i$, we have $\tilde{d}_X(X_1, X_2) = \max_{\vec{x} \in X_2, \vec{x'} \in X_1}(d_X(\vec{x}, \vec{x'})) = \left|x_i - x'_i\right|$, where x_i and x'_i do not depend on the choice of $\vec{x} \in X_1$ and $\vec{x'} \in X_2$. Fixing $\vec{x_0} := [\vec{0}]_{\rho_i}$, we get $\Phi(X_1)(X_2) := \tilde{d}_X(X_1, X_2) - \tilde{d}_X([\vec{0}]_{\rho_i}, X_2) = \left|x_i - x'_i\right| - \left|0 - x'_i\right|$. Here each function $\Phi(X_1)$ is in one-to-one correspondence with a real number x_i, which is the same for all $\vec{x} \in X_1$. A simpler way to define an isometry would be $\Psi : X/\rho_i \to \mathbb{R}$ as $\Psi([x]_{\rho_i}) = x_i$, which is defined correctly since x_i is the same for all elements of $[x]_{\rho_i}$. This function is an isometry w.r.t. the distance $|\cdot|$ of $\mathbb{R}$ since $\tilde{d}_X([\vec{x}]_{\rho_i}, [\vec{x'}]_{\rho_i}) = \left|x_i - x'_i\right|$ for all $\vec{x}, \vec{x'} \in X$.

Coming up with an isometry Φ for a given component ρ and operating with smooth derivatives in resulting Banach space can be difficult in practice. Let us provide a simpler way of constructing Banach spaces, which is well suitable for databases. The following three lemmas give us the basic combinators for statements about derivative sensitivity, reducing the task of finding the derivative sensitivity of a particular function with respect to a particular norm on its domain, to a series of tasks from basic calculus.

Lemma 3.64. *Let $f : \mathbb{R}^n \to \mathbb{R}$, and let $\mathbb{R}^n$ be equipped with the norm ℓ_p. Then $\|df_{\vec{x}}\|$ is the ℓ_q-norm of the gradient vector $\nabla f(\vec{x})$, where $q = \frac{p}{p-1}$ (if $p = 1$ then $q = \infty$ and* vice versa*).*

Proof. Let $\nabla f(\vec{x}) = (a_i)_{i=1}^n$. Assuming $a_i \neq 0$ for all i (otherwise remove the indices i for which $a_i = 0$ from the summations containing a_i):

$$\begin{aligned}
|df_{\vec{x}}(\vec{y})| &= |\nabla f(\vec{x}) \cdot \vec{y}| \leq \sum_{i=1}^{n} |a_i||y_i| \\
&= \sum |a_i|^{\frac{p}{p-1}} \cdot \frac{|y_i|}{|a_i|^{\frac{1}{p-1}}} \\
&\leq \left(\sum |a_i|^{\frac{p}{p-1}}\right)\left(\frac{\sum |y_i|^p}{\sum |a_i|^{\frac{p}{p-1}}}\right)^{\frac{1}{p}} \\
&= \left(\sum |a_i|^{\frac{p}{p-1}}\right)^{\frac{p-1}{p}} \left(\sum |y_i|^p\right)^{\frac{1}{p}} \\
&= \|\nabla f(\vec{x})\|_q \cdot \|\vec{y}\|_p
\end{aligned}$$

for all $\vec{y} \in X$. The second inequality used here is the weighted power means inequality with exponents 1 and p. Equality is achievable (and not only for $\vec{y} = 0$): for example, by taking $y_i = |a_i|^{\frac{1}{p-1}}$. Thus $\|\nabla f(\vec{x})\|_q$ is the smallest value of c such that for all $\vec{y}$, $|df_{\vec{x}}(\vec{y})| \leq c \cdot \|\vec{y}\|_p$, i.e. it is the operator norm $\|df_{\vec{x}}\|$.

The cases $p = 1$ and $p = \infty$ can be achieved as limits of the general case. □

The ℓ_q-norm is the *dual norm* of the ℓ_p-norm; we denote q by $\mathsf{dual}(p)$.

Lemma 3.65. *Let $f : \mathbb{R}^n \to \mathbb{R}$ have the derivative sensitivity g with respect to the norm N. Let $a \cdot N$ denote the scaling of the output of the norm N by $a \in \mathbb{R}_+$. Then f has the derivative sensitivity g/a with respect to $a \cdot N$.*

Proof. The derivative sensitivity of f at $\vec{x}$ is the operator norm of a particular linear operator $df_{\vec{x}}$. It is equal to the minimal possible c, such that for all vectors $\vec{y}$, the absolute value of of $df_{\vec{x}}(\vec{y}) \in \mathbb{R}$ is at most c times larger than the norm $\|\vec{y}\|_N$. If we replace N with $a \cdot N$, then the norm $\|\vec{y}\|_{a \cdot N}$ is increased by a times. Hence we may now reduce c by a times and still have the inequality. □

Lemma 3.66. *(a) Let $(V_1, \| \|_{V_1})$ and $(V_2, \| \|_{V_2})$ be Banach spaces. Let $V = V_1 \times V_2$. Let for all $(v_1, v_2) \in V$,*

$$\|(v_1, v_2)\|_V = \|(\|v_1\|_{V_1}, \|v_2\|_{V_2})\|_p$$

Then $(V, \| \|_V)$ is a Banach space.
(b) Suppose furthermore that a function $f : V \to \mathbb{R}$ is differentiable at each point of V. Fix a point $v = (v_1, v_2) \in V$. Let $g : V_1 \to \mathbb{R}$ be such that $g(x_1) = f(x_1, v_2)$ and $h : V_2 \to \mathbb{R}$ be such that $h(x_2) = f(v_1, x_2)$. Let $c_1 = \mathrm{DS}_g(v_1)$ and $c_2 = \mathrm{DS}_h(v_2)$. Then $\mathrm{DS}_f(v) = \|(c_1, c_2)\|_q$ where $\| \|_q$ is the dual norm of $\| \|_p$.

Proof. (a) We first prove that $(V, \| \|_V)$ is a normed vector space. We prove only the triangle inequality. The rest of the properties of norm are easy to check.

$$\begin{aligned}
&\|(v_1, v_2) + (v_1', v_2')\|_V = \|(v_1 + v_1', v_2 + v_2')\|_V = \|(\|v_1 + v_1'\|_{V_1}, \|v_2 + v_2'\|_{V_2})\|_p \leq \\
&\leq \|(\|v_1\|_{V_1} + \|v_1'\|_{V_1}, \|v_2\|_{V_2} + \|v_2'\|_{V_2})\|_p \leq \\
&\leq \|(\|v_1\|_{V_1}, \|v_2\|_{V_2})\|_p + \|(\|v_1'\|_{V_1}, \|v_2'\|_{V_2})\|_p = \\
&= \|(v_1, v_2)\|_V + \|(v_1', v_2')\|_V
\end{aligned}$$

The first inequality uses the triangle inequalities of $\| \|_{V_1}$ and $\| \|_{V_2}$ and the monotonicity of $\| \|_p$ in the absolute values of the coordinates of its argument vector. The second inequality uses the triangle inequality of $\| \|_p$.

Thus $(V, \| \|_V)$ is a normed vector space. We now prove that it is a Banach space. Consider a Cauchy sequence $\{x_n\}$ in V. Then

$$\forall \epsilon > 0.\ \exists N \in \mathbb{N}.\ \forall m, n > N.\ \|x_m - x_n\|_V < \epsilon$$

Let $x_n = (y_n, z_n)$ where $y_n \in V_1$ and $z_n \in V_2$. Note that

$$\|y_m - y_n\|_{V_1} = \|(y_m - y_n, 0)\|_V \leq \|(y_m - y_n, z_m - z_n)\|_V = \|x_m - x_n\|_V$$

Thus

$$\forall \epsilon > 0.\ \exists N \in \mathbb{N}.\ \forall m, n > N.\ \|y_m - y_n\|_{V_1} < \epsilon$$

i.e. $\{y_n\}$ is a Cauchy sequence in V_1. Because V_1 is a Banach space, there exists $y \in V_1$ such that

$$\lim_{n \to \infty} \|y_n - y\|_{V_1} = 0$$

Similarly, we get that there exists $z \in V_2$ such that

$$\lim_{n \to \infty} \|z_n - z\|_{V_2} = 0$$

Let $x = (y, z)$. Note that

$$\|x_n - x\|_V = \|(y_n - y, z_n - z)\|_V = \|(\|y_n - y\|_{V_1}, \|z_n - z\|_{V_2})\|_p$$

Then, because $\| \|_p$ is continuous,

$$\lim_{n \to \infty} \|x_n - x\|_V = \|(\lim_{n \to \infty} \|y_n - y\|_{V_1}, \lim_{n \to \infty} \|z_n - z\|_{V_2})\|_p = \|(0, 0)\|_p = 0$$

Thus V is a Banach space.
(b)

$$c_1 = \|dg_{v_1}\|$$

$$c_2 = \|dh_{v_2}\|$$

Note that

$$\begin{aligned}
&\lim_{x_1 \to 0_{V_1}} \frac{|g(v_1 + x_1) - g(v_1) - df_v(x_1, 0)|}{\|x_1\|_{V_1}} = \\
&= \lim_{x_1 \to 0_{V_1}} \frac{|f(v_1 + x_1, v_2) - f(v_1, v_2) - df_v(x_1, 0)|}{\|(x_1, 0)\|_V} = \\
&= \lim_{(x_1,0) \to 0_V} \frac{|f(v + (x_1, 0)) - f(v) - df_v(x_1, 0)|}{\|(x_1, 0)\|_V} = \\
&= \lim_{x \to 0_V} \frac{|f(v + x) - f(v) - df_v(x)|}{\|x\|_V} = 0
\end{aligned}$$

The last equality holds by the definition of Fréchet derivative. The equality before that holds because the limit on the right-hand side exists. Then, again by the definition of Fréchet derivative, we get that the linear map that maps x_1 to $df_v(x_1, 0)$, is dg_{v_1}. Thus $df_v(x_1, 0) = dg_{v_1}(x_1)$. Similarly, we get $df_v(0, x_2) = dh_{v_2}(x_2)$. Now

$$\begin{aligned}
&|df_v(x_1, x_2)| = |df_v(x_1, 0) + df_v(0, x_2)| = \\
&= |dg_{v_1}(x_1) + dh_{v_2}(x_2)| \leq |dg_{v_1}(x_1)| + |dh_{v_2}(x_2)| \leq \\
&\leq c_1\|x_1\|_{V_1} + c_2\|x_2\|_{V_2} \leq \|(c_1, c_2)\|_q \cdot \|(\|x_1\|_{V_1}, \|x_2\|_{V_2})\|_p = \\
&= \|(c_1, c_2)\|_q \cdot \|(x_1, x_2)\|_V
\end{aligned}$$

The last inequality follows from the weighted power means inequality, similarly to the proof of Lemma 3.64. Equality is also achievable: because $c_1 = \|dg_{v_1}\|$ and $c_2 = \|dh_{v_2}\|$, there exist x_1 and x_2 that achieve equality in the second inequality. Then scale x_1 and x_2 by constants such that $\|x_1\|_{V_1}$ and $\|x_2\|_{V_2}$ (which scale by the same constants) achieve equality in the third inequality. To achieve equality in the first inequality, we may further need to multiply x_1 and/or x_2 by -1. Thus $\|df_v\| = \|(c_1, c_2)\|_q$. □

The following example demonstrates how the Lemmata 3.64, 3.65, 3.66 can be used to construct a suitable norm and compute derivative sensitivity in the corresponding Banach space.

Example 3.8. Consider the example of computing differentially privately the time that a ship takes to reach the port. This time can be expressed as

$$f(x, y, v) = \frac{\sqrt{x^2 + y^2}}{v}$$

$$f : \mathbb{R}^3 \to \mathbb{R}$$

where (x, y) are the coordinates of the ship (with the port at $(0, 0)$) and v is the speed of the ship. For differential privacy, we need to define distances on $\mathbb{R}^3$ and $\mathbb{R}$. Because we want to use Fréchet derivatives to compute the sensitivities, we instead define norms, which then also induce distances. For $\mathbb{R}$ it is natural to use the absolute value norm but for $\mathbb{R}^3$ there are more choices.

First, let us consider the ℓ_1-distance $\|\Delta x, \Delta y, \Delta v\|_1 = |\Delta x| + |\Delta y| + |\Delta v|$. Then moving the ship by geographical distance s in a direction parallel or perpendicular to its velocity, changes the whole input by ℓ_1-distance s. But moving the ship in any other direction changes the whole input by ℓ_1-distance more than s. This is unnatural. We would like the change not to depend on the direction.

Consider the ℓ_2-distance $\|\Delta x, \Delta y, \Delta v\|_2 = \sqrt{|\Delta x|^2 + |\Delta y|^2 + |\Delta v|^2}$. Then moving the ship by geographical distance s always changes the whole input by ℓ_2-distance s, regardless of direction. If we change the speed of the ship by u (either up or down) then the whole input changes by ℓ_2-distance u. If we simultaneously move the ship by distance s and change its speed by u however, and numerically $s = u$ (ignoring the units) then the whole input changes by ℓ_2-distance $\sqrt{2} \cdot s$ instead of the more natural $2s$.

Thus it is better to combine ℓ_1- and ℓ_2-distances. We first combine the change in the coordinates of the ship using ℓ_1-norm, then combine the result with the change in speed using ℓ_2-norm. Thus

$$||(\Delta x, \Delta y, \Delta v)|| = ||(||(\Delta x, \Delta y)||_2, \Delta v)||_1 = \sqrt{(\Delta x)^2 + (\Delta y)^2} + |\Delta v|$$

This still has a problem. Suppose that $x = y = 2000$ km, $v = 20$ km/h. Then changing v to 10 km/h changes the whole input by the same distance as changing y to 2010 km. But the former change seems much more important in most cases than the latter. Thus we scale the geographical change and the speed change by different constants before combining them:

$$||(\Delta x, \Delta y, \Delta v)|| = a\sqrt{(\Delta x)^2 + (\Delta y)^2} + b|\Delta v| = ||(||(a\Delta x, a\Delta y)||_2, b\Delta v)||_1$$

This norm should now be good enough.

Let us now compute the derivatives. We first compute ordinary partial derivatives, i.e. with respect to the absolute value norm.

$$\frac{\partial f}{\partial x} = \frac{2x}{2v\sqrt{x^2 + y^2}} = \frac{x}{v\sqrt{x^2 + y^2}}$$

$$\frac{\partial f}{\partial y} = \frac{2y}{2v\sqrt{x^2 + y^2}} = \frac{y}{v\sqrt{x^2 + y^2}}$$

$$\frac{\partial f}{\partial v} = -\frac{\sqrt{x^2 + y^2}}{v^2}$$

Now we compute the derivatives w.r.t. the scaled one-dimensional norms, i.e. x is considered not as an element of the Banach space $(\mathbb{R}, |\ |)$ but as an element of the Banach space $(\mathbb{R}, ||\ ||_x)$ where $||\Delta x||_x = a|\Delta x|$. The operator norm of the Fréchet derivative in $(\mathbb{R}, ||\ ||_x)$ (which is also the derivative sensitivity) is then

$$\mathrm{DS}_{f^{y,v}}(x) = ||df_x^{y,v}|| = \left|\frac{\partial f}{a\partial x}\right| = \frac{|x|}{a|v|\sqrt{x^2 + y^2}}$$

where $f^{y,v}$ is the one-variable function defined by $f^{y,v}(x) = f(x, y, v)$, i.e. y and v are considered as constants. Similarly, we compute

$$\mathrm{DS}_{f^{x,v}}(y) = ||df_y^{x,v}|| = \left|\frac{\partial f}{a\partial y}\right| = \frac{|y|}{a|v|\sqrt{x^2 + y^2}}$$

$$\mathrm{DS}_{f^{x,y}}(v) = ||df_v^{x,y}|| = \left|\frac{\partial f}{b\partial v}\right| = \frac{\sqrt{x^2 + y^2}}{bv^2}$$

where $f^{x,v}$ and $f^{x,y}$ are the one-variable functions defined by $f^{x,v}(y) = f(x, y, v)$ and $f^{x,y}(v) = f(x, y, v)$.

Now we use Lemma 3.66 to combine the Banach spaces $(\mathbb{R}, ||\ ||_x)$ and $(\mathbb{R}, ||\ ||_y)$ into the Banach space $(\mathbb{R}^2, ||\ ||_{xy})$ where

$$||(\Delta x, \Delta y)||_{xy} = ||(||\Delta x||_x, ||\Delta y||_y)||_2 = ||(a\Delta x, a\Delta y)||_2$$

We get

$$||df_{(x,y)}^v|| = \mathrm{DS}_{f^v}(x, y) = ||(\mathrm{DS}_{f^{y,v}}(x), \mathrm{DS}_{f^{x,v}}(y))||_2 = \frac{\sqrt{2}}{a|v|}$$

Now we again use Lemma 3.66 to combine the Banach spaces $(\mathbb{R}^2, ||\ ||_{xy})$ and $(\mathbb{R}, ||\ ||_v)$ into the Banach space $(\mathbb{R}^2, ||\ ||_{xyv})$ where

$$||(\Delta x, \Delta y, \Delta v)||_{xyz} = a\sqrt{(\Delta x)^2 + (\Delta y)^2} + b|\Delta v| = ||(||(a\Delta x, a\Delta y)||_2, b\Delta v)||_1$$

We get

$$||df_{(x,y,v)}|| = \mathrm{DS}_f(x, y, v) = ||(\mathrm{DS}_{f^v}(x, y), \mathrm{DS}_{f^{x,y}}(v))||_\infty = \max\left(\frac{\sqrt{2}}{a|v|}, \frac{\sqrt{x^2 + y^2}}{bv^2}\right)$$

Thus we have found not only the derivative sensitivity of f (DS_f) but also its derivative sensitivities w.r.t. components ($\mathrm{DS}_{f^{y,v}}, \mathrm{DS}_{f^{x,v}}, \mathrm{DS}_{f^{x,y}}, \mathrm{DS}_{f^v}$). The sensitivities w.r.t. components depend on other components but this is not a problem because when computing $f(x, y, v)$ differentially privately, even w.r.t. a component, we need to use all of x, y, v anyway, so why not use all this information for computing the noise level too.

Database as a Banach space. Let us have a database with a number of tables. The schema for each table is fixed. We see that database as a point in some Banach space (thus the distance between databases is the norm of their difference), where each cell in each table corresponds to a dimension of that space. As dimensions of Banach spaces are fixed (indeed, they are vector spaces $\mathbb{R}^n$ with some extra structure), the number of rows in each table is also fixed, and each row can be seen to have a public *identity* (similarly to [56]). In this work, we consider the following *composite* norms and seminorms as possible norms for databases:

Definition 3.57 (composite seminorm)**.** Let $\|\cdot\|_N$ be a seminorm of the vector space $\mathbb{R}^n$. It is a *composite seminorm* if one of the following holds for all $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$:

- There exists $i \in [n]$, such that $\|\vec{x}\|_N = |x_i|$. Such seminorm *uses the variable* x_i.
- There exists a composite seminorm $\|\cdot\|_M$ and $a \in \mathbb{R}^+$, such that $\|\vec{x}\|_N = a \cdot \|\vec{x}\|_M$. The seminorm $\|\cdot\|_N$ *uses the same variables* as $\|\cdot\|_M$.
- There exist composite seminorms $\|\cdot\|_{M_1}, \ldots, \|\cdot\|_{M_k}$ and $p \in [1, \infty]$, s.t. $\|\vec{x}\|_N = \|\|\vec{x}\|_{M_1}, \ldots, \|\vec{x}\|_{M_k}\|_p$. The seminorm $\|\cdot\|_N$ uses the union of the variables used by all $\|\cdot\|_{M_i}$.

Let $\mathsf{vars}(N)$ be the set of variables used by $\|\cdot\|_N$.

We normally define the norms for rows of each table using the constructions allowed in Def. 3.57. We then state that the norm of the table is some ℓ_p-norm of the vector of the norms of its rows (last item of Def. 3.57) and the norm of the database is some ℓ_p-norm of the vector of the norms of its tables (in this work, we usually assume that it is ℓ_1-norm). Note that the notion of *seminorm* is used only for building blocks of composite ℓ_p-norms, and a seminorm constructed as in Def. 3.57 becomes a norm if $\mathsf{vars}(N) = \{x_1, \ldots, x_n\}$. Lemmata 3.64, 3.65, 3.66 cover all cases considered in Definition 3.57, thus providing a way to compute derivative sensitivity w.r.t. a composite norm.

In order to compute derivative sensitivity w.r.t. a database, we need to take into account that the query operates on a cross product of tables. The cells of this cross product table are correlated in a certain way, so computing derivative sensitivity for the database is more complicated than computing it for a single table. Let us show how this can be handled.

Suppose we have a database of n tables. Thus we can have a Banach space for each table with a certain norm. Let $T_i = R_i^{n_i}$ be the Banach space for the i^{th} table, where n_i is the number of rows in the i^{th} table and R_i is the Banach space for its row.

The input contains a tuple of n tables, which is an element of $D = T_1 \times \cdots \times T_n$. We can make D a Banach space by combining the norms of T_i using the ℓ_1-norm.

To make a query on the database we want to join those n tables. Consider an input $(t_1, \ldots, t_n) \in D$. Let $t = t_1 \times \cdots \times t_n$ be the cross product of tables. First, let us assume that the n joined tables are distinct, i.e. no table is used more than once. Each row of the cross product is an element of $R = R_1 \times \cdots \times R_n$, thus t is an element of $T = R^N$ where $N = n_1 \cdots n_n$.

The query contains an aggregating function $f : T \to \mathbb{R}$. We can consider $t \in T$ as one large table whose number of rows is the product of the numbers of rows of tables $t_i \in T_i$ and number of columns is the sum of the numbers of columns of tables $t_i \in T_i$. We know how to compute the derivative sensitivity of f w.r.t. the components of t specified by a subset of rows and a subset of columns of t.

Suppose we want to compute the derivative sensitivity of f w.r.t. a row r of t_i. We can compute the sensitivity w.r.t. the following component of t: the subset of rows is the set of rows affected by r, i.e. $t_r = t_1 \times \cdots \times t_{i-1} \times \{r\} \times t_{i+1} \times \cdots \times t_n$, the subset of columns is the set of columns corresponding to

table t_i. Because changing r by distance d changes each row of t_r by distance d, we must combine the rows of t_r by ℓ_∞-distance to ensure that t_r also changes by distance d.

Suppose we now want to compute the sensitivity of f w.r.t. a subset s of rows of t_i. Then the subset of rows affected by s, is $t_s = t_1 \times \cdots \times t_{i-1} \times s \times t_{i+1} \times \cdots \times t_n$. Each row in t_s is affected by exactly one row in s. Let $s = r_1, \ldots, r_k$ and let $t_s = \bigcup_{j=1}^{k} t_{r_j}$ where t_{r_j} is the subset of rows affected by r_j. The t_{r_j} are disjoint. Let $t_{r_j} = \{u_{j1}, \ldots, u_{jm_j}\}$. Then the norm for u_{jm} is the same as the norm for r_j, with the additional columns having zero norm, i.e. not included into the norm. The norm for t_{r_j} is computed by combining the norms for u_{jm} using ℓ_∞-norm. The norm for t_s is then computed by combining the norms for t_{r_j} using the norm that combined the norms of the rows of t_i, i.e. ℓ_{p_i}. If s is the entire set of all rows of t_i, we get derivative sensitivity w.r.t. the table t_i.

We have shown how to compute the derivative sensitivity w.r.t. each table $t_1, \ldots, t_n$ separately. If each table is used only once, since the norm of the database is defined as the ℓ_1-norm of tables, by Lemma 3.64 we need to take the maximum of corresponding derivative sensitivities. If some table is used multiple times, these copies should be combined using ℓ_∞-norm, so we have to sum up the derivative sensitivities of these copies.

3.3.8.3 Smoothing. To achieve differential privacy, we need to find a smooth upper bound on the derivative sensitivity. When differential privacy is required only w.r.t. a component then the sensitivity w.r.t. to the component only needs to be smoothed for changes in that component (changes in other components are allowed to change the smooth upper bound without smoothness restriction). After a smooth upper bound has been found, we can use Theorem 3.62 to compute f differentially privately.

Suppose we have a function $f : \mathbb{R}^n \to \mathbb{R}$ and we want to find its β-smooth upper bound.

A differentiable function $f : \mathbb{R} \to \mathbb{R}$ is β-smooth if $\left|\frac{f'(x)}{f(x)}\right| \le \beta$.

If DS_f exists then $f : X \to \mathbb{R}$ is β-smooth if $\frac{\mathrm{DS}_f(x)}{|f(x)|} \le \beta$.

We will now show how to compute β-smooth bounds on f and DS_f for certain functions f. First, let us state and prove some helpful lemmas.

Lemma 3.67 (a part of Lemma 2.3 of [52])**.** *Let $f : X \to \mathbb{R}$. For $\beta > 0$, a β-smooth upper bound on f is*

$$g_{f,\beta} = \max_{x' \in X}(f(x) \cdot e^{-\beta \cdot d(x,x')}) \ .$$

Lemma 3.68. *Let X_i for $i \in \{1, \ldots, n\}$ be Banach spaces, $f_i : X_i \to \mathbb{R}$. Let $x = (x_1, \ldots, x_n)$, and let $f(x_1, \ldots, x_n) = \|f_1(x_1), \ldots, f_n(x_n)\|_p$. Then $\frac{\partial f}{\partial x_i}(x) = \frac{\partial f_i}{\partial x_i}(x_i) \cdot \left(\frac{f_i(x_i)}{f(x)}\right)^{p-1} \le \frac{\partial f_i}{\partial x_i}(x_i)$.*

Proof. Let $y_i = f_i(x_i)$ and $y = (y_1, \ldots, y_n)$. We have

$$\frac{\partial f}{\partial x_i}(x) = \frac{\partial f}{\partial y_i}(y) \cdot \frac{\partial f_i}{\partial x_i}(x_i) = \left(\frac{y_i}{\|y\|_p}\right)^{p-1} \cdot \frac{\partial f_i}{\partial x_i}(x_i) \ .$$

Since $\frac{y_i}{\|y\|_p} = \frac{f_i(x_i)}{f(x)} = \frac{f_i(x_i)}{\|(f_i(x_i))_{i=1}^n\|_p}$, we have $\frac{f_i(x_i)}{f(x)} \le 1$, and hence also $\left(\frac{f_i(x_i)}{f(x)}\right)^{p-1} \le 1$, getting $\frac{\partial f}{\partial x_i}(x) \le \frac{\partial f_i}{\partial x_i}(x_i)$. □

Lemma 3.69. *Let X_i for $i \in \{1, \ldots, n\}$ be Banach spaces. Let $x = (x_1, \ldots, x_n)$, and let $f(x) = \|f_1(x), \ldots, f_n(x)\|_p$. Then $\frac{\partial f}{\partial x_i}(x) = \sum_{j=1}^{n} \left(\frac{f_j(x)}{f(x)}\right)^{p-1} \cdot \frac{\partial f_j}{\partial x_i}(x)$. This can be upper bounded as:*

1. $\sum_{j=1}^{n} \frac{\partial f_j}{\partial x_i}(x)$;
2. $\max_{j=1}^{n} \frac{f(x)}{f_j(x)} \cdot \frac{\partial f_j}{\partial x_i}(x)$.

Proof. Let $y_j = f_j(x), z = \sum_{j=1}^n y_j^p$. We have

$$\begin{aligned}\frac{\partial f}{\partial x_i}(x) &= \frac{\partial f}{\partial z}(z) \cdot \sum_{j=1}^n \left(\frac{\partial f_j(x)^p}{\partial y_j}(y_j) \cdot \frac{\partial f_j}{\partial x_i}(x) \right) \\ &= \sum_{j=1}^n \left(\frac{y_j}{\|y\|_p} \right)^{p-1} \cdot \frac{\partial f_j}{\partial x_i}(x) \\ &= \sum_{j=1}^n \left(\frac{f_j(x)}{f(x)} \right)^{p-1} \cdot \frac{\partial f_j}{\partial x_i}(x) \ .\end{aligned}$$

As in the proof of Lemma 3.68, $\left(\frac{y_j}{\|y\|_p} \right)^{p-1} = \left(\frac{f_j(x)}{f(x)} \right)^{p-1} \leq 1$. We get $\frac{\partial f}{\partial x_i}(x) \leq \sum_{j=1}^n \frac{\partial f_j}{\partial x_i}(x)$. We can proceed with the inequality in another way.

$$\begin{aligned}\sum_{j=1}^n \left(\frac{f_j(x)}{f(x)} \right)^{p-1} \cdot \frac{\partial f_j}{\partial x_i}(x) &= \frac{\sum_{j=1}^n f_j(x)^{p-1} \cdot \frac{\partial f_j}{\partial x_i}(x)}{f(x)^{p-1}} \\ &= \frac{\sum_{j=1}^n f_j(x)^p \cdot \frac{\partial f_j}{\partial x_i}(x) \cdot \frac{1}{f_j(x)}}{f(x)^{p-1}} \\ &\leq \max_{j=1}^n \left(\frac{1}{f_j(x)} \cdot \frac{\partial f_j}{\partial x_i}(x) \right) \cdot \frac{\sum_{j=1}^n f_j(x)^p}{f(x)^{p-1}} \\ &= \max_{j=1}^n \left(\frac{1}{f_j(x)} \cdot \frac{\partial f_j}{\partial x_i}(x) \right) \cdot \frac{f(x)^p}{f(x)^{p-1}} \\ &= \max_{j=1}^n \left(\frac{f(x)}{f_j(x)} \cdot \frac{\partial f_j}{\partial x_i}(x) \right) \ . \quad \square\end{aligned}$$

Lemma 3.70. *Let $f(x) : \mathbb{R} \to \mathbb{R}$ be β_f-smooth, and let $g(x) : \mathbb{R} \to \mathbb{R}$ be β_g-smooth.*

1. *If $f(x), g(x) > 0$, then $f(x) + g(x)$ is $\max(\beta_f, \beta_g)$-smooth;*
2. *$f(x) \cdot g(x)$ is $\beta_f + \beta_g$-smooth;*
3. *$f(x) \,/\, g(x)$ is $\beta_f + \beta_g$-smooth.*

Proof. We have:

1. $\left| \frac{(f(x)+g(x))'}{f(x)+g(x)} \right| = \frac{|f'(x)+g'(x)|}{|f(x)|+|g(x)|} \leq \frac{|f'(x)|+|g'(x)|}{|f(x)|+|g(x)|}$
   $\leq \max\left(\left| \frac{f'(x)}{f(x)} \right|, \left| \frac{g'(x)}{g(x)} \right| \right) \leq \max(\beta_f, \beta_g).$
2. $\left| \frac{(f(x)\cdot g(x))'}{f(x)\cdot g(x)} \right| = \left| \frac{f'(x)\cdot g(x)+f(x)\cdot g'(x)}{f(x)\cdot g(x)} \right|$
   $\leq \left| \frac{f'(x)}{f(x)} \right| + \left| \frac{g'(x)}{g(x)} \right| \leq \beta_f + \beta_g.$
3. $\left| \frac{(f(x)/g(x))'}{f(x)/g(x)} \right| = \left| \frac{f'(x)\cdot g(x)-f(x)\cdot g'(x)}{g(x)^2} \cdot \frac{g(x)}{f(x)} \right|$
   $= \left| \frac{f'(x)}{f(x)} - \frac{g'(x)}{g(x)} \right| \leq \left| \frac{f'(x)}{f(x)} \right| + \left| \frac{g'(x)}{g(x)} \right| \leq \beta_f + \beta_g.$

$\square$

Lemma 3.71. *Let X_i for $i \in \{1, \ldots, n\}$ be Banach spaces, and let $X = \prod_{i=1}^n X_i$. Let $f_i : X_i \to \mathbb{R}$ be β_i-smooth. Then, $f(x_1, \ldots, x_n) = \|f_1(x_1), \ldots, f_n(x_n)\|_p$ is $\|(\beta_i)_{i=1}^n\|_p$-smooth as well as $\max_{i=1}^n(\beta_i)$-smooth, where the norm of X is the $\ell_{\mathrm{dual}(p)}$-combination of the norms of all X_i.*

Proof. Let $X = \prod_{i=1}^{n} X_i$ and $x = (x_1, \ldots, x_n)$. Let $\beta = \max_i \beta_i$. By Lemma 3.68, an upper bound on $\frac{\partial f}{\partial x_i}(x)$ is $c_i(x) = f_i'(x_i)$. We have

$$|c_i(x)| = \left|f_i'(x_i)\right| \leq \mathrm{DS}_{f_i}(x_i) = |f_i(x_i)| \cdot \frac{\mathrm{DS}_{f_i}(x_i)}{|f_i(x_i)|} \leq |f_i(x_i)| \cdot \beta_i \ .$$

By Lemma 3.64 and Lemma 3.66, the derivative sensitivity of f in $(X, \ell_{\frac{p}{p-1}})$ is

$$\mathrm{DS}_f(x) = \|(c_1(x), \ldots, c_n(x))\|_p$$

Using inequality $|f_i(x_i)| \leq |f(x)|$, we get

$$\frac{\mathrm{DS}_f(x)}{|f(x)|} \leq \frac{\left\|(|f_i(x_i)| \cdot \beta_i)_{i=1}^n\right\|_p}{|f(x)|} \leq \frac{|f(x)| \cdot \left\|(\beta_i)_{i=1}^n\right\|_p}{|f(x)|} \leq \|(\beta_i)_{i=1}^n\|_p \ .$$

On the other hand, using inequality $\beta_i \leq \beta$, we get

$$\frac{\mathrm{DS}_f(x)}{|f(x)|} \leq \frac{\left\|(|f_i(x_i)| \cdot \beta)_{i=1}^n\right\|_p}{|f(x)|} \leq \frac{\beta \cdot \left\|(|f_i(x_i)|)_{i=1}^n\right\|_p}{|f(x)|} = \beta$$

□

Lemma 3.72. *Let X_i for $i \in \{1, \ldots, n\}$ be Banach spaces, $X = \prod_{i=1}^{n} X_i$. Let $f_i : X \to \mathbb{R}$ be β_i^j-smooth for X_j. Let $x = (x_1, \ldots, x_n)$. Then, $f(x) = \|f_1(x), \ldots, f_n(x)\|_p$ is $\|(\max_j \beta_i^j)_{i=1}^n\|_p$-smooth.*

Proof. Let $X = \prod_{i=1}^{n} X_i$ and $x = (x_1, \ldots, x_n)$. By Lemma 3.69, an upper bound on $\frac{\partial f}{\partial x_i}(x)$ is $c_i(x) = \max_j^n \frac{f(x)}{f_j(x)} \cdot \frac{\partial f_j}{\partial x_i}(x)$. We have

$$|c_i(x)| = \left|\max_j \frac{f(x)}{f_j(x)} \cdot \frac{\partial f_j}{\partial x_i}(x)\right| \leq |f(x)| \cdot \max_j^n \left|\frac{\partial f_j}{\partial x_i}(x) \cdot \frac{1}{f_j(x)}\right| \leq |f(x)| \cdot \max_j \beta_i^j \ .$$

By Lemma 3.64 and Lemma 3.66, the derivative sensitivity of f in $(X, \ell_{\mathsf{dual}(p)})$ is

$$\mathrm{DS}_f(x) = \|(c_1(x), \ldots, c_n(x))\|_p$$

We get

$$\frac{\mathrm{DS}_f(x)}{|f(x)|} \leq \frac{|f(x)| \cdot \left\|\left(\max_j \beta_i^j\right)_{i=1}^n\right\|_p}{|f(x)|} \leq \|(\max_j \beta_i^j)_{i=1}^n\|_p.$$

□

We are now ready to find smooth upper bounds for certain functions.

Power function. Let $f(x) = x^r, r \in \mathbb{R}_+, x > 0$. We have

$$\frac{f'(x)}{f(x)} = \frac{rx^{r-1}}{x^r} = \frac{r}{x} \ ; \qquad \left|\frac{r}{x}\right| \leq \beta \Leftrightarrow x \geq \frac{|r|}{\beta} \ .$$

For $x \leq \frac{r}{\beta}$, the function $f'(x)$ achieves its maximum at the point $\frac{r}{\beta}$. By Lemma 3.67, a β-smooth upper bound on f is

$$\mathrm{UB}_f(x) = \begin{cases} x^r & \text{if } x \geq \frac{r}{\beta} \\ e^{\beta x - r}\left(\frac{r}{\beta}\right)^r & \text{otherwise} \end{cases}$$

If $r \geq 1$, we may also find a smooth upper bound on the derivative sensitivity DS_f of f. We have

$$\mathrm{DS}_f(x) = |f'(x)| = |r|x^{r-1} \ .$$

A β-smooth upper bound on DS_f is

$$\mathrm{UB}_{\mathrm{DS}_f}(x) = \begin{cases} rx^{r-1} & \text{if } x \geq \frac{r-1}{\beta} \\ re^{\beta x - (r-1)}\left(\frac{r-1}{\beta}\right)^{r-1} & \text{otherwise} \end{cases}$$

Exponent. Let $f(x) = e^{rx}, r \in \mathbb{R}, x \in \mathbb{R}$. We have $\mathrm{DS}_f(x) = |f'(x)| = |r|e^{rx}$, hence:

- $\left|\frac{f'(x)}{f(x)}\right| = \frac{re^{rx}}{e^{rx}} = r$; $\left|\frac{f''(x)}{f(x)}\right| = \frac{r^2e^{rx}}{re^{rx}} = r$.

Thus both f and DS_f are β-smooth if $|r| \leq \beta$.

Sigmoid. Consider the (sigmoid) function $\sigma(x) = \frac{e^{\alpha x}}{e^{\alpha x}+1}$. This function can be viewed as a continuous approximation of the indicator function $I_{\mathbb{R}_+} : \mathbb{R} \to \{0, 1\}$, which is less precise for values close to 0, and the error decreases when α increases. We have:

- $\sigma'(x) = \frac{\alpha e^{\alpha x}}{(e^{\alpha x}+1)^2}$; $\sigma''(x) = \frac{\alpha^2 e^{\alpha x}(e^{\alpha x}-1)}{(e^{\alpha x}+1)^3}$;
- $\left|\frac{\sigma'(x)}{\sigma(x)}\right| = \left|\alpha \cdot \frac{1}{e^{\alpha x}+1}\right| \leq \alpha$; $\left|\frac{\sigma''(x)}{\sigma'(x)}\right| = \left|\alpha \cdot \frac{e^{\alpha x}-1}{e^{\alpha x}+1}\right| \leq \alpha$.

Thus both $\sigma(x)$ and $DS_{\sigma(x)} = |\sigma'(x)|$ are α-smooth. If we want less DP noise, we should decrease α, which in turn makes the sigmoid itself less precise, so there is a tradeoff.

Tauoid. Consider the function $\tau(x) = \frac{2}{e^{-\alpha x}+e^{\alpha x}}$ (let us call it a *tauoid*). This function can be viewed as a continuous approximation of the indicator function $I_{\{0\}} : \mathbb{R} \to \{0, 1\}$, which works similarly to a sigmoid. We have:

$$\begin{aligned}
\tau'(x) &= -\frac{2\alpha(e^{\alpha x} - e^{-\alpha x})}{(e^{-\alpha x} + e^{\alpha x})^2} \\
&= \frac{2\alpha(e^{-\alpha x} - e^{\alpha x})}{(e^{-\alpha x} + e^{\alpha x})^2} = \frac{2\alpha e^{\alpha x}(1 - e^{2\alpha x})}{(1 + e^{2\alpha x})^2} \\
\left|\frac{\tau'(x)}{\tau(x)}\right| &= \frac{|\alpha| \cdot |e^{-\alpha x} - e^{\alpha x}|}{e^{-\alpha x} + e^{\alpha x}} \leq |\alpha| \\
|\tau'(x)| &\leq \frac{2|\alpha|e^{\alpha x}}{1 + e^{2\alpha x}} = \frac{2|\alpha|}{e^{-\alpha x} + e^{\alpha x}} \\
&= |\alpha|\tau(x) =: \mathrm{UB}_{\mathrm{DS}_\tau}(x) \\
\mathrm{UB}'_{\mathrm{DS}_\tau}(x) &= |\alpha|\tau'(x) \\
\left|\frac{\mathrm{UB}'_{\mathrm{DS}_\tau}(x)}{\mathrm{UB}_{\mathrm{DS}_\tau}(x)}\right| &= \left|\frac{\tau'(x)}{\tau(x)}\right| \leq |\alpha| \ .
\end{aligned}$$

Thus both τ itself and $\mathrm{UB}_{\mathrm{DS}_\tau}$, an upper bound on its derivative sensitivity, are α-smooth.

An ℓ_p-norm. Consider the function $f(x) = \|x\|_p = \left(\sum x_i^p\right)^{1/p}$, $x \in \mathbb{R}^n$, $x = (x_1, \ldots, x_n)$. We have

$$\frac{\partial f}{\partial x_i}(x) = \frac{p x_i^{p-1}}{p\left(\sum x_i^p\right)^{(p-1)/p}} = \left(\frac{x_i^p}{\sum x_i^p}\right)^{(p-1)/p} \ .$$

By Lemma 3.64, the derivative sensitivity of f in $(\mathbb{R}^n, \ell_p)$ is

$$\mathrm{DS}_f(x) = \left(\sum \frac{x_i^p}{\sum x_i^p}\right)^{\frac{p-1}{p}} = 1 \ .$$

This is constant and thus β-smooth for all β. The function f itself is β-smooth if $\frac{1}{\|x\|_p} \leq \beta$, i.e. if $\|x\|_p \geq \frac{1}{\beta}$. By Lemma 3.67, a β-smooth upper bound on f is

$$\mathrm{UB}_f(x) = \begin{cases} \|x\|_p & \text{if } \|x\|_p \geq \frac{1}{\beta} \\ \frac{e^{\beta\|x\|_p - 1}}{\beta} & \text{otherwise} \end{cases}$$

This also holds for $p = \infty$.

The ℓ_∞-norm. Let $f(x) = \|x\|_\infty = \max_i |x_i|$. We have

$$\frac{\partial f}{\partial x_i}(x) = \begin{cases} 1 & \text{if } i = \mathrm{argmax}_j \left|x_j\right| \\ \text{undefined} & \text{if } \mathrm{argmax}_j \left|x_j\right| \text{ is not unique} \\ 0 & \text{otherwise} \end{cases}$$

The derivative sensitivity of f in $(\mathbb{R}^n, \ell_\infty)$ is

$$\mathrm{DS}_f(x) = \begin{cases} 1 & \text{if } \operatorname{argmax}_j \left|x_j\right| \text{ is unique} \\ \text{undefined} & \text{if } \operatorname{argmax}_j \left|x_j\right| \text{ is not unique} \end{cases}$$

Because we are interested in upper bounds on the derivative sensitivity, we define

$$\mathrm{DS}_f(x_0) := \limsup_{x \to x_0} \mathrm{DS}_f(x) = 1$$

for those x_0 for which $\mathrm{DS}_f(x_0)$ is undefined. Thus $\mathrm{DS}_f(x) = 1$, which is constant and β-smooth for all β. The smooth upper bound on the function f itself can be found similarly to the ℓ_p-norm case.

The following constructions are considered in Fig. 41.

Product. Let $f : \prod_{i=1}^n X_i \to \mathbb{R}, f(x_1, \ldots, x_n) = \prod_{i=1}^n f_i(x_i)$ where X_i are Banach spaces. Let $X = \prod_{i=1}^n X_i$ and $x = (x_1, \ldots, x_n)$. First, suppose that variables x_i are independent. We have $\frac{\partial f}{\partial x_i}(x) = \prod_{i \neq j=1}^n f_j(x_j) \cdot f_i'(x_i)$, and $\left|\frac{\partial f}{\partial x_i}(x) \cdot \frac{1}{f(x)}\right| = \left|\frac{f_i'(x_i)}{f_i(x_i)}\right|$, hence:

- If $\left|\frac{f_i'(x_i)}{f_i(x_i)}\right| \leq \beta$, then f is β-smooth w.r.t. x_i.

- By Lemmas 3.64 and 3.66,

$$\|df_x\| = \left\| \left(\prod_{i \neq j=1}^{n} f_j(x_j) \cdot f_i'(x_i) \right)_{i=1}^n \right\|_{\frac{p}{p-1}}$$

  in (X, ℓ_p), so we have $\frac{\|df_x\|}{|f(x)|} = \frac{\|df_x\|}{\left|\prod_{i=1}^n f_i(x_i)\right|} = \left\| \left(\frac{f_i'(x_i)}{f_i(x_i)} \right)_{i=1}^n \right\|_{\frac{p}{p-1}} \leq \|(\beta_i)_{i=1}^n\|_{\frac{p}{p-1}}$, where β_i is the smoothness of f_i. Hence, if f_i is β-smooth w.r.t. x_i for all i, then f is β-smooth in (X, ℓ_1) and $n\beta$-smooth in (X, ℓ_∞).

The derivative sensitivity of f w.r.t. x_i is $c_i(x) = \mathrm{DS}_{f_i}(x_i) \cdot \left|\frac{f(x)}{f_i(x_i)}\right|$. The derivative sensitivity of f in (X, ℓ_p) is, by Lemma 3.66, $\mathrm{DS}_f(x) = \|(c_1(x), \ldots, c_n(x))\|_{\frac{p}{p-1}} = \left\| \left(\frac{\mathrm{DS}_{f_i}(x_i)}{|f_i(x_i)|} \right)_{i=1}^n \right\|_{\frac{p}{p-1}} \cdot |f(x)|$.

We have $c_i(x) = \mathrm{DS}_{f_i}(x_i) \cdot \left|\frac{f(x)}{f_i(x_i)}\right| = \mathrm{DS}_{f_i}(x_i) \cdot \prod_{j \neq i} \left|f_j(x_j)\right|$. Since $\prod_{j \neq i} \left|f_j(x_j)\right|$ does not depend on x_i and $\mathrm{DS}_{f_i}(x_i) \geq 0$, by Lemma 3.70, if DS_{f_i} is β-smooth in X_i then $c_i(x)$ is also β-smooth in X_i. Similarly, if $f_j(x_j)$ is β-smooth, then $c_i(x)$ is also β-smooth in X_j. Hence, if f_i and DS_{f_i} are β-smooth for all i, by Lemma 3.72, DS_f is β-smooth in (X, ℓ_1) and $n\beta$-smooth in (X, ℓ_∞). If DS_{f_i} are not all β-smooth then we can use their β-smooth upper bounds when computing c_i. Then we get a β-smooth upper bound on DS_f instead of the actual DS_f.

We may also consider the case where the variables x_i are fully dependent, i.e. equal (the case where they are partially dependent is currently not considered). Consider a function $f(x) = g(x) \cdot h(x)$ where $g, h : X \to \mathbb{R}_+$ and X is a Banach space. We have

$$\mathrm{DS}_f(x) = g(x) \cdot \mathrm{DS}_h(x) + h(x) \cdot \mathrm{DS}_g(x) \ .$$

By Lemma 3.70, if g is β_g-smooth, h is β_h-smooth, DS_g is $\beta_{g'}$-smooth, and DS_h is $\beta_{h'}$-smooth, then DS_f is $\max(\beta_g + \beta_{h'}, \beta_h + \beta_{g'})$-smooth. The function f itself is $(\beta_g + \beta_h)$-smooth.

Sum. Let $f : \prod_{i=1}^n X_i \to \mathbb{R}, f(x_1, \ldots, x_n) = \sum_{i=1}^n f_i(x_i)$ where X_i are Banach spaces. Let $X = \prod_{i=1}^n X_i$ and $x = (x_1, \ldots, x_n)$. First, suppose that the variables x_i are independent. The derivative sensitivity of f w.r.t. x_i is $\mathrm{DS}_{f_i}(x_i)$. By Lemmas 3.64 and 3.66, the derivative sensitivity of f in (X, ℓ_p) is $\mathrm{DS}_f(x) = \|\mathrm{DS}_{f_1}(x_1), \ldots, \mathrm{DS}_{f_n}(x_n)\|_{\frac{p}{p-1}}$.

- Let $f_i \geq 0$ for all $i \in \{1, \ldots, n\}$ (or $f_i \leq 0$ for all $i \in \{1, \ldots, n\}$) and β_i-smooth w.r.t. X_i. Now we have $|f(x)| = \sum_{i=1}^n |f_i(x_i)| = \||f_i(x_i)|_{i=1}^n\|_1$. By Lemma 3.71, $f(x)$ is $\beta := \max_i(\beta_i)$-sensitive in (X, ℓ_p). We do not get a good bound in the case when f_i may have different signs, since then $f_i(x)$ may cancel each other out and make $f(x)$ arbitrarily small even if $|f_i(x)|$ are large.

- Let DS_{f_i} be β_i-smooth for $i \in \{1, \ldots, n\}$. By Lemma 3.71, DS_f is $\|(\beta_i)_{i=1}^n\|_{\frac{p}{p-1}}$-smooth in (X, ℓ_p), and if all DS_{f_i} are β-smooth, then DS_f is also β-smooth.

Consider the case where x_i are equal: $f(x) = \sum_{i=1}^n g_i(x)$ where $g_i : X \to \mathbb{R}$ and X is a Banach space. Then

$$\mathrm{DS}_f(x) = \sum_{i=1}^n \mathrm{DS}_{g_i}(x)$$

By Lemma 3.70, if all DS_{g_i} are β-smooth then DS_f is β-smooth. If all g_i are non-negative and β-smooth then f is β-smooth.

Min / max. Let $f : \prod_{i=1}^n X_i \to \mathbb{R}, f(x_1, \ldots, x_n) = \min_{i=1}^n f_i(x_i)$ where X_i are Banach spaces (the case with max instead of min is similar). Let $X = \prod_{i=1}^n X_i$ and $x = (x_1, \ldots, x_n)$. Let the variables x_i be independent.

If for all i, f_i is β-smooth in X_i then f is β-smooth in (X, ℓ_p). The same holds with max or sum (with non-negative f_i) or $\ell_{p'}$-norm instead of min.

The derivative sensitivity of f w.r.t. x_i is $\mathrm{DS}_{f_i}(x_i)$ if $i = \operatorname{argmin} f_i(x_i)$ and 0 otherwise. The derivative sensitivity of f in (X, ℓ_p) is $\mathrm{DS}_f(x) = \mathrm{DS}_{f_i}(x_i)$ where $i = \operatorname{argmin} f_i(x_i)$. In general, DS_f is discontinuous at points where $\operatorname{argmin} f_i(x_i)$ is not unique.

A possible valid β-smooth (in (X, ℓ_p)) upper bound on DS_f is $\max c_i(x_i)$ where c_i is a β-smooth upper bound on DS_{f_i}.

Norm scaling. Let $f : X \to \mathbb{R}$ in the Banach space $(X, \|\cdot\|)$. Scaling the norm by a scales the derivative $f'(x)$ by $\frac{1}{a}$ while keeping the value of $f(x)$ the same. Hence, if f is β-smooth in $(X, \|\cdot\|)$ then it is $\frac{\beta}{a}$-smooth in $(X, a \cdot \|\cdot\|)$.

Let $c(x)$ be a β-smooth upper bound on the derivative sensitivity of f at x in $(X, \|\cdot\|)$. Then $\frac{c(x)}{a}$ is a $\frac{\beta}{a}$-smooth upper bound on the derivative sensitivity of f at x in $(X, a \cdot \|\cdot\|)$ by Lemma 3.65.

Sensitivity w.r.t. a larger norm. Let $f : X \to \mathbb{R}$ in the Banach space $(X, \|\cdot\|_N)$. Let $\|\cdot\|_M \geq \|\cdot\|_N$.

If f is β-smooth in $(X, \|\cdot\|_N)$, then $f(x) \leq e^{\beta\|x-x'\|_N} \cdot f(x') \leq e^{\beta\|x-x'\|_M} \cdot f(x')$ for all $x, x' \in X$, so f is also β-smooth in $(X, \|\cdot\|_M)$. The same holds about any function that is β-smooth in $(X, \|\cdot\|_N)$, including a β-smooth upper bound on the derivative sensitivity of f.

Let us show that the derivative sensitivity of f w.r.t. $\|\cdot\|_N$ is a valid upper bound on the derivative sensitivity of f w.r.t. $\|\cdot\|_M$. First, note that $\|\cdot\|_{\mathrm{dual}(N)} \geq \|\cdot\|_{\mathrm{dual}(M)}$. Indeed, by definition of a dual norm, $\|T\|_{\mathrm{dual}(M)} = \sup\{T(x) \mid \|x\|_M \leq 1\}$ for an operator T from the dual space $X \to \mathbb{R}$ of X. Since $\|x\|_N \leq \|x\|_M$, we have $\forall x : \{T(x) \mid \|x\|_N \leq 1\} \supseteq \{T(x) \mid \|x\|_M \leq 1\}$. Hence, $\|T\|_{\mathrm{dual}(N)} = \sup\{T(x) \mid \|x\|_N \leq 1\} \geq \sup\{T(x) \mid \|x\|_M \leq 1\} = \|T\|_{\mathrm{dual}(M)}$.

By definition, we have $\mathrm{DS}_f(x) = \|df_x\|_{\mathrm{dual}(N)}$, where df_x is the Fréchet derivative of f at x. Since $\|\cdot\|_{\mathrm{dual}(N)} \geq \|\cdot\|_{\mathrm{dual}(M)}$, we have $\|df_x\|_{\mathrm{dual}(N)} \geq \|df_x\|_{\mathrm{dual}(M)}$.

Composition with a real function. Let $f(x) = h(g(x)), x \in X$ where $g : X \to \mathbb{R}, h : \mathbb{R} \to \mathbb{R}$ and X is a Banach space.

$$\mathrm{DS}_f(x) = |h'(g(x))| \cdot \mathrm{DS}_g(x)$$

$$\frac{\mathrm{DS}_f(x)}{|f(x)|} = \frac{|h'(g(x))|}{|h(g(x))|} \cdot \mathrm{DS}_g(x)$$

Suppose that h is β_h-smooth and $\mathrm{DS}_g(x) \leq B$ for all x. Then f is $\beta_h B$-smooth. We have

$$\mathrm{DS}_{\mathrm{DS}_f}(x) = |h''(g(x))|(\mathrm{DS}_g(x))^2 + |h'(g(x))| \cdot \mathrm{DS}_{\mathrm{DS}_g}(x) \ ,$$

$$\frac{\mathrm{DS}_{\mathrm{DS}_f}(x)}{\mathrm{DS}_f(x)} = \frac{|h''(g(x))|}{|h'(g(x))|} \cdot \mathrm{DS}_g(x) + \frac{\mathrm{DS}_{\mathrm{DS}_g}(x)}{\mathrm{DS}_g(x)} \ .$$

By Lemma 3.70, if h' is $\beta_{h'}$-smooth, DS_g is $\beta_{g'}$-smooth, and $\mathrm{DS}_g(x) \leq B$ for all x then DS_f is $(\beta_{h'} B + \beta_{g'})$-smooth.

Example 3.9. Consider an example of computing differentially privately the time it takes for the next ship to reach the port. This time can be expressed as

$$f(x_1, y_1, v_1, \ldots, x_n, y_n, v_n) = \min_{i=1}^{n} \frac{\sqrt{x_i^2 + y_i^2}}{v_i}$$

$$f : \mathbb{R}^{3n} \to \mathbb{R}$$

where (x_i, y_i) are the coordinates of the i^{th} ship (with the port at $(0, 0)$) and v_i is its speed.

Note that v_i is in the power -1 and we do not know how to find the smooth derivative sensitivity of the function $f_{v,i}(v_i) = v_i^{-1}$ (we only know how to do it for power functions with exponent ≥ 1). Let us define $w_i = \zeta \ln v_i$. The coefficient ζ is used to control the distance by which the whole input vector changes if $\ln v_i$ is changed by 1. Similarly, we add a coefficient to the geographical coordinates: $s_i = \alpha x_i, t_i = \alpha y_i$. Then we consider $(s_1, t_1, w_1, \ldots, s_n, t_n, w_n)$ as an element of the Banach space $(\mathbb{R}^{3n}, \| \|)$ where

$$\|(s_1, t_1, w_1, \ldots, s_n, t_n, w_n)\| = \|(\|(\|(s_1, t_1)\|_2, w_1)\|_1, \ldots, \|(\|(s_n, t_n)\|_2, w_n)\|_1)\|_p$$

Then $v_i = e^{w_i/\zeta}, x_i = \frac{s_i}{\alpha}, y_i = \frac{t_i}{\alpha}$ and

$$g(s_1, t_1, w_1, \ldots, s_n, t_n, w_n) = \frac{1}{\alpha} \min_{i=1}^{n} \frac{\sqrt{s_i^2 + t_i^2}}{e^{w_i/\zeta}}$$

Now the derivative sensitivity of $g_{w,i}(w_i) = e^{-w_i/\zeta}$ is

$$\mathrm{DS}_{g_{w,i}}(w_i) = \frac{1}{\zeta} e^{-w_i/\zeta}$$

which is $\frac{1}{\zeta}$-smooth. The function $g_{w,i}$ itself is also $\frac{1}{\zeta}$-smooth.

The derivative sensitivity of $g_{st,i}(s_i, t_i) = \sqrt{s_i^2 + t_i^2}$ in $(\mathbb{R}^2, \ell_2)$ is

$$\mathrm{DS}_{g_{st,i}}(s_i, t_i) = 1$$

which is β-smooth for all β. The function $g_{st,i}$ is $\frac{1}{\zeta}$-smooth if $\frac{1}{\sqrt{s_i^2+t_i^2}} \leq \frac{1}{\zeta}$, i.e. if $\sqrt{s_i^2 + t_i^2} \geq \zeta$. A $\frac{1}{\zeta}$-smooth upper bound on $g_{st,i}$ is

$$\hat{g}_{st,i}(s_i, t_i) = \begin{cases} \sqrt{s_i^2 + t_i^2} & \text{if } \sqrt{s_i^2 + t_i^2} \geq \zeta \\ \zeta e^{\frac{\sqrt{s_i^2+t_i^2}}{\zeta} - 1} & \text{otherwise} \end{cases}$$

An upper bound on the derivative sensitivity of $g_i(s_i, t_i, w_i) = \frac{\sqrt{s_i^2+t_i^2}}{e^{w_i/\zeta}}$ is

$$c_{g_i}(s_i, t_i, w_i) = \left\|(\mathrm{DS}_{g_{st,i}}(s_i, t_i) \cdot g_{w,i}(w_i), \mathrm{DS}_{g_{w,i}}(w_i) \cdot \hat{g}_{st,i}(s_i, t_i))\right\|_\infty =$$

$$= \left\|\left(1 \cdot e^{-w_i/\zeta}, \frac{1}{\zeta} e^{-w_i/\zeta} \cdot \hat{g}_{st,i}(s_i, t_i)\right)\right\|_\infty = \frac{\max\left(1, \frac{\hat{g}_{st,i}(s_i,t_i)}{\zeta}\right)}{e^{w_i/\zeta}}$$

and it is $\frac{1}{\zeta}$-smooth because $\mathrm{DS}_{g_{w,i}}(w_i)$, $\mathrm{DS}_{g_{st,i}}$, $g_{w,i}(w_i)$, and $\hat{g}_{st,i}(s_i, t_i))$ are $\frac{1}{\zeta}$-smooth.

A $\frac{1}{\zeta}$-smooth upper bound on DS_g is

$$c(u) = \frac{1}{\alpha} \max_i c_{g_i}(s_i, t_i, w_i) = \frac{1}{\alpha} \max_i \frac{\max\left(1, \frac{\hat{g}_{st,i}(s_i,t_i))}{\zeta}\right)}{e^{w_i/\zeta}}$$

where $u = (s_1, t_1, w_1, \ldots, s_n, t_n, w_n)$.

Now we can use Theorem 3.62 to compute an ϵ-differentially private version of g:

$$h(u) = g(u) + \frac{c(u)}{b} \cdot \eta$$

$$\epsilon = (\gamma + 1)(b + \frac{1}{\zeta})$$

$$\gamma > 1, b > 0, \eta \sim \mathit{GenCauchy}(\gamma)$$

To compute an ϵ-differentially private version of f, we first transform $(x_1, y_1, v_1, \ldots, x_n, y_n, v_n)$ into u and then compute $h(u)$.

Smooth derivative sensitivity w.r.t. multiple tables. To achieve differential privacy, we need a β-smooth upper bound on derivative sensitivity. For this, we first compute an upper bound on the sensitivity w.r.t. a single table that is β-smooth w.r.t. all the necessary tables. We compute this for each necessary table and then take the maximum. Maximum preserves β-smoothness and because we are using ℓ_1-norm, we get an upper bound on the sensitivity w.r.t. all needed tables. It remains to show how to ensure that the upper bound on derivative sensitivity is β-smooth w.r.t. all the necessary tables (and not only the table w.r.t. which the sensitivity was computed).

To compute an upper bound on the sensitivity w.r.t. *a single table* that is β-smooth w.r.t. *all the tables*, we force the sensitivity w.r.t. the other tables to zero. If the other tables are not sensitive (i.e. do not contain any attributes included in the database norm), we use the exact values of subexpressions instead of their smooth upper bounds. For the other tables that are sensitive (i.e. contain at least one attribute included in the database norm), we still compute β-smooth upper bounds on the values of subexpressions (but their derivative sensitivities are 0).

If a table is used more than once then we compute the derivative sensitivity w.r.t. each copy of the table separately and add the result together because changing a row in the table is equivalent to changing a row in each copy of the table. A problem occurs with smoothing. If the sensitivity w.r.t. one copy is β-smooth w.r.t. each copy separately then changing a row in one copy by distance d changes the sensitivity by up to $e^{\beta d}$ times. Changing it in all m copies changes the sensitivity by up to $e^{m\beta d}$ times. Thus the sensitivity w.r.t. the initial table (instead of one of its copies) is $m\beta$-smooth instead of β-smooth.

To achieve β-smoothness w.r.t. each initial table, we require the sensitivities to be (for all i) $\frac{\beta}{m_i}$-smooth w.r.t. each copy of table i where m_i is the number of times the table i is used. Our analysis does not directly support different values of β for different (copies of) tables, thus we need to scale the β at appropriate points during the analysis. Scaling at the end is not optimal because then we would achieve $\frac{\beta}{m}$-smoothness w.r.t. each copy of each table, where $m = \max_i m_i$, even for those tables that are used less than m times. Thus we scale at the beginning (at leaf nodes) but this requires also using a scaled norm for global sensitivity because global sensitivity is used for computing smooth sensitivity at some intermediate nodes (compose versions of exponent e^x, sigmoid $\frac{e^{ax}}{e^{ax}+1}$, and tauoid $\frac{2}{e^{ax}+e^{-ax}}$).

In the analyses where we also need a non-scaled global sensitivity (e.g. used for smoothing the combined sensitivity described in Sec. 3.3.9), we just compute the global sensitivity twice w.r.t. two different norms.

Optimizing smoothness parameter. Theorem 3.62 tells how to use derivative sensitivity to achieve differential privacy in a Banach space. We prove a stronger version of this theorem that allows us to choose the best value of β (the one giving the smallest noise level) according to the actual input, instead of having to guess it in advance.

Theorem 3.73. *Let $\gamma, b, \beta \in \mathbb{R}_+$, $\gamma > 1$. Let $f : X \to \mathbb{R}$ where X is a Banach space. Let $\mathcal{I}$ be a (possibly infinite) set of indices. For all $I \in \mathcal{I}$, let $\epsilon = (\gamma + 1)(b_I + \beta_I)$ and c_I be a β_I-smooth upper bound on DS_f. Let η be a random variable distributed according to GenCauchy(γ). Then $g(\vec{x}) = f(\vec{x}) + h(\vec{x}) \cdot \eta$, where $h(\vec{x}) = \min_{I \in \mathcal{I}} \frac{c_I(\vec{x})}{b_I}$ (provided that the minimum exists), is ϵ-differentially private.*

Proof. Let $\vec{x}, \vec{x'} \in X$. Denote $L = \|\vec{x'} - \vec{x}\|$. We have to show that $d_{\mathrm{dp}}(g(\vec{x}), g(\vec{x'})) \leq \epsilon L$.

Let $n \in \mathbb{N}$ be arbitrary. Let $\vec{v}_0 = \vec{x}$, $\vec{v}_n = \vec{x'}$, and $\vec{v}_i = \frac{n-i}{n} \cdot \vec{x} + \frac{i}{n} \cdot \vec{x'}$. Consider $i \in \{1, \ldots, n\}$. There exist $I, J \in \mathcal{I}$ such that $h(\vec{v}_{i-1}) = \frac{c_I(\vec{v}_{i-1})}{b_I}$ and $h(\vec{v}_i) = \frac{c_J(\vec{v}_i)}{b_J}$. Note that $\frac{c_I(\vec{v}_{i-1})}{b_I} \leq \frac{c_J(\vec{v}_{i-1})}{b_J}$ and $\frac{c_I(\vec{v}_i)}{b_I} \geq \frac{c_J(\vec{v}_i)}{b_J}$. Because c_I and c_J are continuous, there exists a point $\vec{u}$ on the (closed) segment connecting $\vec{v}_{i-1}$ to $\vec{v}_i$ where $\frac{c_I(\vec{u})}{b_I} = \frac{c_J(\vec{u})}{b_J}$. Let $K, M \in \{I, J\}$ be such that $\beta_K = \max(\beta_I, \beta_J)$, $\beta_M = \min(\beta_I, \beta_J)$, and $\{K, M\} = \{I, J\}$. Then

$$\left|\ln \frac{h(\vec{v}_i)}{h(\vec{v}_{i-1})}\right| \leq \left|\ln \frac{h(\vec{v}_i)}{h(\vec{u})}\right| + \left|\ln \frac{h(\vec{u})}{h(\vec{v}_{i-1})}\right| \leq \beta_J \|\vec{v_i} - \vec{u}\| + \beta_I \|\vec{u} - \vec{v_{i-1}}\| \leq \beta_K L/n \ .$$

The mean value theorem for Banach spaces and Fréchet derivative gives

$$|f(\vec{v}_i) - f(\vec{v}_{i-1})| \leq \|df_{\vec{t_i}}\| \cdot \|\vec{v}_i - \vec{v}_{i-1}\| \leq c_K(\vec{t_i}) \cdot \frac{L}{n} \leq e^{\beta_K L/n} \cdot c_K(\vec{v}_{i-1}) \cdot \frac{L}{n}$$

for some point $\vec{t_i}$ on the segment connecting $\vec{v}_{i-1}$ to $\vec{v}_i$. Here we also used the fact that c_K is an upper bound on DS_f.

If $h(\vec{v}_{i-1}) = \frac{c_K(\vec{v}_{i-1})}{b_K}$ then $c_K(\vec{v}_{i-1}) = b_K \cdot h(\vec{v}_{i-1})$. Otherwise

$$h(\vec{v}_{i-1}) = \frac{c_M(\vec{v}_{i-1})}{b_M} \geq \frac{e^{-\beta_M \|\vec{u} - \vec{v}_{i-1}\|} c_M(\vec{u})}{b_M} = \frac{e^{-\beta_M \|\vec{u} - \vec{v}_{i-1}\|} c_K(\vec{u})}{b_K} \geq \frac{e^{-\beta_M \|\vec{u} - \vec{v}_{i-1}\|} e^{-\beta_K \|\vec{v}_{i-1} - \vec{u}\|} c_K(\vec{v_{i-1}})}{b_K} = \frac{e^{-(\beta_I + \beta_J) L/n} c_K(\vec{v_{i-1}})}{b_K} \ .$$

In both cases, $c_K(\vec{v_{i-1}}) \leq e^{(\beta_I + \beta_J) L/n} \cdot b_K \cdot h(\vec{v}_{i-1})$. Thus

$$|f(\vec{v}_i) - f(\vec{v}_{i-1})| \leq e^{(\beta_I + \beta_J + \beta_K) L/n} \cdot b_K \cdot h(\vec{v}_{i-1}) \cdot \frac{L}{n}$$

and

$$d_{\mathrm{dp}}(g(\vec{v}_{i-1}), g(\vec{v}_i)) = d_{\mathrm{dp}}(f(\vec{v}_{i-1}) + h(\vec{v}_{i-1}) \cdot \eta, f(\vec{v}_i) + h(\vec{v}_i) \cdot \eta) \leq (\gamma + 1) \left(\frac{|f(\vec{v}_i) - f(\vec{v}_{i-1})|}{|h(\vec{v}_{i-1})|} + \frac{\beta_K L}{n} \right) \leq$$

$$(\gamma + 1) \left(b_K \cdot e^{(\beta_I + \beta_J + \beta_K) L/n} \cdot \frac{L}{n} + \frac{\beta_K L}{n} \right) = (\gamma + 1)(b_K e^{(\beta_I + \beta_J + \beta_K) L/n} + \beta_K) L/n \leq (\gamma + 1)(b_K e^{3\beta L/n} + \beta_K) L/n$$

where $\beta = \sup_{I \in \mathcal{I}} \beta_I$. This supremum exists because for all $I \in \mathcal{I}, \beta_I = \frac{\epsilon}{\gamma+1} - b_I \leq \frac{\epsilon}{\gamma+1}$. Now

$$d_{\mathrm{dp}}(g(\vec{x}), g(\vec{x'})) \leq \sum_{i=1}^{n} d_{\mathrm{dp}}(g(\vec{v}_{i-1}), g(\vec{v}_i)) \leq L/n \cdot (\gamma + 1) \sum_{i=1}^{n} \max_{K \in \mathcal{I}} b_K e^{3\beta L/n} + \beta_K \leq$$

$$L/n \cdot (\gamma + 1) \sum_{i=1}^{n} \max_{K \in \mathcal{I}} e^{3\beta L/n} (b_K + \beta_K) = L/n \cdot e^{3\beta L/n} \sum_{i=1}^{n} \max_{K \in \mathcal{I}} (\gamma + 1)(b_K + \beta_K) =$$

$$L/n \cdot e^{3\beta L/n} \sum_{i=1}^{n} \max_{K \in \mathcal{I}} \epsilon = e^{3\beta L/n} \epsilon L \ .$$

This inequality holds for any $n \in \mathbb{N}$. If $n \to \infty$ then $e^{3\beta L/n} \to 1$ and we obtain the inequality that we had to show. □

3.3.8.4 Derivative Sensitivity of SQL Queries. We now describe how the theory of derivative sensitivity for Banach spaces can be applied to SQL queries, computing the smooth upper bounds of their derivative sensitivities, such that an appropriate amount of noise may be added to turn them differentially private. Our theory deals with functions that return a numeric value, so the query should return a single output. We do not support DISTINCT queries, as we do not know how to continuously approximate

Continuous function	Approximated condition	name
$\frac{e^{\alpha x}}{e^{\alpha x}+1}$	$x \geq 0$	sigmoid
$\frac{2}{e^{-\alpha x}+e^{\alpha x}}$	$x = 0$	tauoid

Table 9: Continuous Approximations of Step Functions

efficiently a function that removes repeating elements from a list of arguments. We consider queries of the form

SELECT *aggr expr* FROM t1 AS s1,...,tn AS sn
WHERE *condition* ,

where:

- *expr* is an expression over table columns, computed as a continuous function.
- *condition* is a boolean expression over predicates $P(x) \in \{x < 0, x = 0\}$, where x is an expression of the same form as *expr*. Since all functions have to be continuous, these predicates are computed using continuous approximations to the step functions, listed in Table. 9.
- *aggr* is one of the operations SUM, COUNT, MIN, MAX.

GROUP BY queries can be simulated by generating for each group a separate query, with a filter selecting that particular group. Hence, we can group either by a public or a discrete attribute to get a finite number of groups.

Continuous approximation of an SQL query. We assume that the expression under SELECT statement is an application of a continuous function to the database. Hence, for a query without a filter, we can directly apply results of Section 3.3.8.3 to find a smooth upper bound on derivative sensitivity,

A filter that *does not depend* on sensitive data can be applied directly to the cross product of the input tables, and we may then proceed with the query without a filter.

A filter that *does depend* on sensitive data is treated as a part of the query. We treat this filter as a continuous function, applied in such a way that the discarded rows would be ignored by the aggregating function. We combine sigmoids and tauoids to obtain the approximated value of the indicator $\sigma(x_i) \in \{0, 1\}$, denoting whether the row x_i satisfies the filter.

Hence, if the filter depends on sensitive data, we have the following set-up:

- There is a set of rows $\{x_1, \ldots, x_m\}$.
- There is a function f_i applied to the row x_i, returning a real number. For different rows, this function may be different, e.g. it may be determined by the public cells of the row.
- There is a filtering function σ_i applied to the row x_i. It returns a real number. It approximates a boolean condition, i.e. its values are mostly near 0 and 1.
- There is an aggregation function applied to a subset of the values $f_1(x_1), \ldots, f_m(x_m)$. Only such $i \in \{1, \ldots, m\}$ are selected, where the condition holds.

To convert the SQL query into a continuous function, the functions f_i and σ_i are combined as follows, depending on the aggregation function:

- SUM. The values 0 do not affect the sum, hence we compute the result as $\sum_{i=1}^{m} f_i(x_i) \cdot \sigma_i(x_i)$.
- COUNT: The values of f_i do not affect the result. We compute the result as $\sum_{i=1}^{m} \sigma(x_i)$, counting all entries for which $\sigma_i(x_i) = 1$. The sensitivity of such query only depends on the sensitivity of σ.

Table 10: Upper Bounds for Uni- and Multivariate functions

$f(x)$	cond.s	g, s.t. $f \le g \sim_{\lvert\cdot\rvert} \beta$	h, s.t. $\mathrm{DS}_f \le'_{\lvert\cdot\rvert} h \sim \beta$
x^r	$r \ge 1$, $x > 0$	$\begin{cases} x^r & \text{if } x \ge \frac{r}{\beta} \\ \mathsf{pw}^r_\beta(x) & \text{oth.} \end{cases}$	$\begin{cases} rx^{r-1} & \text{if } x \ge \frac{r-1}{\beta} \\ r \cdot \mathsf{pw}^{r-1}_\beta(x) & \text{oth.} \end{cases}$

$f(x)$	cond.s	g, s.t. $f \le g \sim_{\lvert\cdot\rvert} \beta$	h, s.t. $\mathrm{DS}_f \le'_{\lvert\cdot\rvert} h \sim \beta$
e^{rx}	$\lvert r\rvert \le \beta$	e^{rx}	$\lvert r\rvert e^{rx}$
c	$c \in \mathbb{R}$	c	0
$\frac{e^{\alpha x}}{e^{\alpha x}+1}$	$\beta \ge \alpha$	$\frac{e^{\alpha x}}{e^{\alpha x}+1}$	$\frac{\alpha e^{\alpha x}}{(e^{\alpha x}+1)^2}$
$\frac{e^{\alpha x}}{e^{\alpha x}+1}$	$\beta < \alpha$	1	$\frac{\alpha e^{\beta x}}{(e^{\beta x}+1)^2}$
$\frac{2e^{\alpha x}}{1+e^{2\alpha x}}$	$\beta \ge \alpha$	$\frac{2e^{\alpha x}}{1+e^{2\alpha x}}$	$\frac{2\lvert\alpha\rvert e^{\alpha x}}{1+e^{2\alpha x}}$

$f(\vec{x})$	g, s.t. $f \le g \sim_{\ell_p} \beta$	h, s.t. $\mathrm{DS}_f \le'_{\ell_p} h \sim \beta$
$\lVert\vec{x}\rVert_p$	$\begin{cases} \lVert\vec{x}\rVert_p & \text{if } \lVert\vec{x}\rVert_p \ge \frac{1}{\beta} \\ \mathsf{pw}^1_\beta(\lVert\vec{x}\rVert_p) & \text{oth.} \end{cases}$	1

Here $\mathsf{pw}^r_\beta(x) = \left(\frac{r}{\beta}\right)^r \cdot e^{\beta x - r}$

- MIN, MAX: If $\sigma_i(x_i)$ is 0, then we need to replace the actual value $f_i(x_i)$ with some large [resp. small] value that would not affect the result of MIN [resp. MAX]. Our conversion of the SQL query proceeds by first defining $\Delta := \mathrm{MAX}(f(x_1), \ldots, f(x_n)) - \mathrm{MIN}(f(x_1), \ldots, f(x_n))$, and then computing the result by applying MIN to the values $f_i(x_i) + (1 - \sigma_i(x_i)) \cdot \Delta$. MAX is computed similarly, changing the first "+" into a "−".

If we know that the compared values are integers and hence $d(x, x') \ge 1$ for $x \ne x'$, we can do better than using sigmoids or tauoids from Table 9, defining precise functions:

- $x > y \iff \min(1, \max(0, x - y))$.
- $x = y \iff 1 - \min(1, \max(0, |x - y|))$.

An advantage of these functions is that they do not lose precision due to addition and multiplication.

For real numbers, we may bound the precision and assume e.g. that $d(x, x') \ge 1/k$ for some $k \ge 1$, which allows to use similar functions. The sensitivity of such comparisons will be k times larger than for integers.

Inferring derivative sensitivities. Let the write-up $\mathrm{DS}_f \le'_N h \sim \beta$ mean that h is a β-smooth upper bound on the derivative sensitivity of f, according to the norm $\|\cdot\|_N$ on the domain of f. For compositions, we also need the upper bounds for the (absolute values of) functions f themselves; let $f \sim_N \beta$ denote that f is β-smooth, and $f \le g \sim_N \beta$ denote that g is a β-smooth upper bound of $|f|$, again according to the norm $\|\cdot\|_N$ on the domain of f. Table 10 lists the smooth upper bounds of some simple uni- and multivariate functions and their derivative sensitivities (using absolute value as the norm on $\mathbb{R}$). This table is a summary of Sec. 3.3.8.3 where these upper bounds are proved. For composite functions, the rules for computing the β-smooth upper bounds are given in Fig. 41.

Query norm vs user-defined norm. The facts and rules in Table 10 and Fig. 41 are in principle sufficient to compute the smooth upper bounds of derivative sensitivity of functions resulting from SQL queries with respect to *all* composite ℓ_p-norms. Still, when we naïvely apply them, we end up finding the sensitivity for a particular norm that is somehow "natural" for the function. In practice, it may happen that we actually need sensitivity w.r.t. some different norm, because the data owner specified so. For example, we know how to compute the sensitivity w.r.t. the norm $\|x_1, x_2\|_1$, but are interested in differential privacy w.r.t. $\|x_1, x_2\|_2$.

$$\frac{\forall i : \mathrm{DS}_{f_i} \leq'_N h_i \sim \beta}{\mathrm{DS}_{\sum_i c_i f_i} \leq'_N \sum_i c_i h_i \sim \beta}(+_D) \qquad \frac{\mathrm{DS}_{f_i} \leq'_{N_i} h_i \sim \beta_i \quad \forall i,j : \mathsf{vars}(N_i) \perp \mathsf{vars}(N_j) \quad \forall i,j : f_i \cdot f_j \geq 0}{\mathrm{DS}_{\sum_{i=1}^k f_i} \leq'_{\ell_p(N_1,\ldots,N_k)} \|h_1,\ldots,h_k\|_{\mathsf{dual}(p)} \sim \|\beta_1,\ldots,\beta_k\|_{\mathsf{dual}(p)}}(+^{\perp}_D)$$

$$\frac{\forall i : f_i \leq g_i \sim_N \beta}{\sum_i c_i f_i \leq \sum_i c_i g_i \sim_N \beta}(+_S) \qquad \frac{f_i \leq g_i \sim_N \beta_i \quad \mathrm{DS}_{f_i} \leq'_N h_i \sim \beta'_i}{\mathrm{DS}_{f_1 \cdot f_2} \leq'_N g_1 \cdot h_2 + g_2 \cdot h_1 \sim \max(\beta_1 + \beta'_2, \beta'_1 + \beta_2)}(*_D)$$

$$\frac{f_i \leq g_i \sim_{N_i} \beta \quad \mathrm{DS}_{f_i} \leq'_{N_i} h_i \sim \beta \quad \mathsf{vars}(N_1) \perp \mathsf{vars}(N_2)}{\mathrm{DS}_{f_1 \cdot f_2} \leq'_{N_1+N_2} g_1 \cdot h_2 + g_2 \cdot h_1 \sim \beta}(*^{\perp}_D)$$

$$\frac{f_i \leq g_i \sim_N \beta_i}{f_1 \cdot f_2 \leq g_1 \cdot g_2 \sim_N \beta_1 + \beta_2}(*_S) \qquad \frac{f_i \leq g_i \sim_{N_i} \beta \quad \mathsf{vars}(f_1) \perp \mathsf{vars}(f_2)}{f_1 \cdot f_2 \leq g_1 \cdot g_2 \sim_{N_1+N_2} \beta}(*^{\perp}_S)$$

$$\frac{\mathrm{DS}_f \leq'_N g \sim \beta \quad N \leq M}{\mathrm{DS}_f \leq'_M g \sim \beta}(\leq_D) \qquad \frac{\mathrm{DS}_f \leq'_N g \sim \beta \quad \forall \vec{x} : \bar{g}(\vec{x}) = g(\vec{x})/a}{\mathrm{DS}_f \leq'_{a \cdot N} \bar{g} \sim \beta/a}(N_D)$$

$$\frac{f \sim_N \beta}{f \sim_{a \cdot N} \beta/a}(N_S) \qquad \frac{\mathrm{DS}_{f_1} \leq'_N h_1 \sim \beta_1 \quad \forall \vec{x} : h_1(\vec{x}) \leq B \quad f'_2 \sim_{|\cdot|} \beta_2}{\mathrm{DS}_{f_2 \circ f_1} \leq'_N |f'_2 \circ f_1| \cdot h_1 \sim \beta_2 B + \beta_1}(\circ_D)$$

$$\frac{f_2 \sim_{|\cdot|} \beta \quad \forall \vec{x} : \mathrm{DS}_{f_1}(\vec{x}) \leq B}{f_2 \circ f_1 \sim_N \beta B}(\circ_S) \qquad \frac{\mathrm{DS}_{f_i} \leq'_{N_i} h_i \sim \beta \quad \forall i,j : \mathsf{vars}(N_i) \perp \mathsf{vars}(N_j)}{\mathrm{DS}_{\min\{f_i,\ldots,f_k\}} \leq'_{\ell_p(N_1,\ldots,N_k)} \max\{h_1,\ldots,h_k\} \sim \beta}(\min^{\perp}_D)$$

$$\frac{f_i \leq g_i \sim_{N_i} \beta \quad \forall i,j : \mathsf{vars}(N_i) \perp \mathsf{vars}(N_j)}{\min\{f_1,\ldots,f_k\} \leq \min\{g_1,\ldots,g_k\} \sim_{\ell_p(N_1,\ldots,N_k)} \beta}(\min^{\perp}_S)$$

Figure 41: Upper Bounds for Composite Functions

Let the *query norm* (denoted N_{qr}) be the norm for which we *can* compute derivative sensitivity. Let the *user-defined norm* (denoted N_{db}) be the norm for which we *want to* compute derivative sensitivity.

Let $N \leq M$, denote that $\|\vec{x}\|_N \leq \|\vec{x}\|_M$ for all $\vec{x}$. If $N_{qr} \leq N_{db}$, then the rule $(\leq_D)$ allows us to use the computed DS_f for N_{qr} also with the norm N_{db}. But if $N_{qr} \not\leq N_{db}$, then we cannot directly use the sensitivity w.r.t. N_{qr}.

According to rule (N_D), if a the upper bound to the derivative sensitivity of the function f is β-smooth according to N_{qr}, then, its $\frac{1}{\alpha}$-times scaled version is $\frac{\beta}{\alpha}$-smooth in according to the norm $\alpha \cdot N_{qr}$ for any $\alpha > 0$. We compute sensitivity w.r.t. such norm $\alpha \cdot N_{qr}$, that $\alpha \cdot N_{qr} \leq N_{db}$. The sensitivity becomes $\frac{\beta}{\alpha}$-smooth instead of β-smooth, which affects the amount of noise required to achieve differential privacy.

Lemma 3.74. *Let $\vec{x} = (x_1,\ldots,x_k) \in \mathbb{R}^k$, $\vec{y} = (y_1,\ldots,y_n) \in \mathbb{R}^n$, $\vec{z} = (z_1,\ldots,z_m) \in \mathbb{R}^m$. If $p \geq q \geq 1$, then*

1. $\|\|\vec{x}\|_q, \|\vec{y}\|_q, z_1,\ldots,z_m\|_p \leq \|\|\vec{x}|\vec{y}\|_q, z_1,\ldots,z_m\|_p$;
2. $\|\|\vec{x}\|_p, \|\vec{y}\|_p, z_1,\ldots,z_m\|_q \geq \|\|\vec{x}|\vec{y}\|_p, z_1,\ldots,z_m\|_q$;

where $\vec{x}|\vec{y}$ denotes concatenation. If $p = q$, then the inequalities become equalities.

Proof. Since $p, q \geq 1$, we may raise both sides of equations to the powers p or q. The main inequalities that we use in the proof are $a^n + b^n \leq (a+b)^n$ for $n \geq 1$, and $a^n + b^n \geq (a+b)^n$ for $n \leq 1$.

$$\begin{aligned}
\|\|\vec{x}\|_q, \|\vec{y}\|_q, z_1,\ldots,z_m\|_p^p &= \left(\sum_{i=1}^k x_i^q\right)^{\frac{p}{q}} + \left(\sum_{i=1}^n y_i^q\right)^{\frac{p}{q}} + \sum_{i=1}^m z_i^p \\
&\leq \left(\sum_{i=1}^k x_i^q + \sum_{i=1}^n y_i^q\right)^{\frac{p}{q}} + \sum_{i=1}^m z_i^p \\
&= \|\vec{x}|\vec{y}\|_q^p + \sum_{i=1}^m z_i^p = \|\|\vec{x}|\vec{y}\|_q, z_1,\ldots,z_m\|_p^p \ .
\end{aligned}$$

$$
\begin{aligned}
\|\|\vec{x}\|_p, \|\vec{y}\|_p, z_1, \ldots, z_m\|_q^q &= \left(\sum_{i=1}^{k} x_i^p\right)^{\frac{q}{p}} + \left(\sum_{i=1}^{n} y_i^p\right)^{\frac{q}{p}} + \sum_{i=1}^{m} z_i^q \\
&\geq \left(\sum_{i=1}^{k} x_i^p + \sum_{i=1}^{n} y_i^p\right)^{\frac{q}{p}} + \sum_{i=1}^{m} z_i^q \\
&= \|\vec{x}\vec{y}\|_p^q + \sum_{i=1}^{m} z_i^q = \|\|\vec{x}\vec{y}\|_p, z_1, \ldots, z_m\|_q^q \ .
\end{aligned}
$$

If $p = q$, then all inequalities in these derivations are equalities. □

Lemma 3.75. *Let N be a composite ℓ_p-norm over $\vec{x} = (x_1, \ldots, x_n)$. Let composite seminorms N' and $V_1, \ldots, V_m$ be such, that $N = N'(V_1, \ldots, V_m)$, and for all $i \in [n]$ let W_i be a seminorm such that $V_i \leq W_i$. Then, $N'(V_1, \ldots, V_m) \leq N'(W_1, \ldots, W_m)$.*

Proof. Let $N = N'(V_1, \ldots, V_m)$. The relation $V_i \leq W_i$ implies $\|x_1, \ldots, x_n\|_{V_i} \leq \|x_1, \ldots, x_n\|_{W_i}$ for all $x_1, \ldots, x_n \in \mathbb{R}^n$. Define a new norm $M = N'(W_1, \ldots, W_m)$. By definition of a composite ℓ_p-norm, we have the three cases for N'.

- If $N' = |x_j|$ for some $j \in [n]$, then $m = 0$, and hence $\|x_1, \ldots, x_n\|_N = \|x_1, \ldots, x_n\|_M = |x_j|$.
- If $N' = \alpha z$, then $m = 1$, and we have $\|x_1, \ldots, x_n\|_N = \alpha\|x_1, \ldots, x_n\|_{V_1}$, and $\|x_1, \ldots, x_n\|_M = \alpha\|x_1, \ldots, x_n\|_{W_1}$, so $N \leq M$.
- If $N' = \|z_1, \ldots, z_m\|_p$, then $\|x_1, \ldots, x_n\|_N = \|\|x_1, \ldots, x_n\|_{V_1}, \ldots, \|x_1, \ldots, x_n\|_{V_m}\|_p \leq \|\|x_1, \ldots, x_n\|_{W_1}, \ldots, \|x_1, \ldots, x_n\|_{W_m}\|_p = \|x_1, \ldots, x_n\|_M$, so $N \leq M$.

In any case, we get $N \leq M$, which is equivalent to $N'(V_1, \ldots, V_m) \leq N'(W_1, \ldots, W_m)$. □

Lemma 3.76. *For all $x \in \mathbb{R}$, $(\alpha_1, \ldots, \alpha_k) \in \mathbb{R}^k$, $(y_1, \ldots, y_m) \in \mathbb{R}^m$: $\|\alpha_1 x, \ldots, \alpha_k x, y_1, \ldots, y_m\|_p = \|\sqrt[p]{\sum_{i=1}^{k} \alpha_i^p}\, x, y_1, \ldots, y_m\|_p$.*

Proof. Since an ℓ_p-norm is defined for $p \geq 1$, we may raise both sides of equation to the power p. We use the definition of ℓ_p-norm and rewrite the term.

$$
\begin{aligned}
&\|\alpha_1 x, \ldots, \alpha_k x, y_1, \ldots, y_m\|_p^p \\
&= \sum_{i=1}^{k} (\alpha_i x)^p + \sum_{i=1}^{m} y_i^p \\
&= \left(\sum_{i=1}^{k} \alpha_i^p\right) x^p + \sum_{i=1}^{m} y_i^p \\
&= \|\sqrt[p]{\sum_{i=1}^{k} \alpha_i^p}\, x, y_1, \ldots, y_m\|_p^p \ .
\end{aligned}
$$

□

Lemma 3.77. *Let $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$. Let N be a composite ℓ_p-norm, defined over variables $\vec{x}$. There exist $0 \leq \alpha_i, \beta_i \in \mathbb{R}$ for $i \in [n]$, such that $\|\alpha_1 x_1, \ldots, \alpha_n x_n\|_p \leq \|x_1, \ldots, x_n\|_N \leq \|\beta_1 x_1, \ldots, \beta_n x_n\|_q$, where:*

- *p is the largest ℓ_p-norm constructor in N;*
- *q is the smallest ℓ_p-norm constructor in N.*

Proof. Without loss of generality, we assume that all scalings in N are applied directly to the variables, as we can always apply the equality $\alpha\|\vec{x}\| = \|\alpha\vec{x}\|$ to push all scalings as deep as possible, directly in front of variables. Let the variable x_i occur k_i times in N, and let α_{ij} be the scaling of the j-th occurrence of x_i. We define α_i and β_i of Lemma 3.77 as follows.

- $\alpha_i = \sqrt[p]{\sum_{j=1}^{k_i} \alpha_{ij}}$.
- $\beta_i = \sqrt[q]{\sum_{j=1}^{k_i} \alpha_{ij}}$.

We prove the first inequality, and the proof would be analogous for the second one. Let $N = \|M_1, \ldots, M_k\|_r$. Since p is the largest ℓ_p-norm used as a term constructor of N, we have $\|M_1, \ldots, M_k\|_r \geq \|M_1, \ldots, M_k\|_p$. Repeat the same procedure with all $M_1, \ldots, M_k$ recursively, substituting all instances of ℓ_r with ℓ_p. By Lemma 3.75, each step of the transformation keeps the resulting norm smaller (or equal). Finally, we are left with a composite ℓ_p-norm N' that only contains $\|\cdot\|_p$ for the same $p \geq 1$ as a term constructor. We can now apply Lemma 3.74 and get a norm of the form $N' = \|\alpha_{11}x_1, \ldots, \alpha_{nk_n}x_n\|_p$, such that $N' \leq N$.

Some variables x_i used by N' may repeat if they were repeating in N before. We may now use Lemma 3.76 to merge repeating variables into one, rewriting

$$\|\alpha_{11}x_1, \ldots, \overbrace{\alpha_{i1}x_i, \ldots, \alpha_{ik_i}x_i}^{k_i}, \ldots, \alpha_{nk_n}x_n\|_p = \|\alpha_{11}x_1, \ldots, \sqrt[p]{\sum_{j=1}^{k_i} \alpha_{ij}}x_i, \ldots, \alpha_{nk_n}x_n\|_p.$$

After doing it for all $i \in [n]$, we get a norm $N'' = \|\alpha_1 x_1, \ldots, \alpha_n x_n\|_p$, which satisfies $N'' \leq N$. □

We show that a suitable α always exists for a composite ℓ_p-norm (Def. 3.57) if N_{db} uses all the variables $x_1, \ldots, x_n$. This assumption is reasonable: any variable that N_{db} does not use is not treated as sensitive, so we may treat it as a constant when computing the sensitivity, reducing the total number of variables. We can find α as follows.

1. Use Lemma 3.77 to get $a_i, b_i \geq 0$, $p, q > 0$ satisfying the conditions $\|a_1x_1, \ldots, a_nx_n\|_q \geq \|x_1, \ldots, x_n\|_{N_{qr}}$ and $\|b_1x_1, \ldots, b_nx_n\|_p \leq \|x_1, \ldots, x_n\|_{N_{db}}$. We have $a \cdot \|x_1, \ldots, x_n\|_p \geq \|a_1x_1, \ldots, a_nx_n\|_p$ for $a = \max_i a_i$, and $b \cdot \|x_1, \ldots, x_n\|_p \leq \|b_1x_1, \ldots, b_nx_n\|_p$ for $b = \min_i b_i$.

   We get $a \cdot \|x_1, \ldots, x_n\|_p \geq \|x_1, \ldots, x_n\|_{N_{qr}}$. Since N_{db} uses all variables $x_1, \ldots, x_n$, we have $b_i \neq 0$ for all i, and hence $b \neq 0$. This allows to write $\|x_1, \ldots, x_n\|_p \leq \frac{1}{b} \cdot \|x_1, \ldots, x_n\|_{N_{db}}$.
2. If $p \leq q$, we have $a \cdot \|x_1, \ldots, x_n\|_q \leq a \cdot \|x_1, \ldots, x_n\|_p$. If $p > q$, we can use equivalence of ℓ_p-norms that gives us $a \cdot \|x_1, \ldots, x_n\|_q \leq n^{1/q-1/p} \cdot a \cdot \|x_1, \ldots, x_n\|_p$. Let $c = (p \leq q)\ ?\ 1\ :\ n^{1/q-1/p}$.
3. We have now come up with scalings a, b, c that satisfy $c \cdot a \cdot \|x_1, \ldots, x_n\|_p \geq \|x_1, \ldots, x_n\|_{N_{qr}}$, and $\|x_1, \ldots, x_n\|_p \leq \frac{1}{b} \cdot \|x_1, \ldots, x_n\|_{N_{db}}$. Putting these inequalities together, we get $c \cdot a \cdot \frac{1}{b} \cdot \|x_1, \ldots, x_n\|_{N_{db}} \geq \|x_1, \ldots, x_n\|_{N_{qr}}$. By construction, we always have $c > 0$. It is possible that $a = 0$ only in the case if $a_i = 0$ for all i, i.e. the query uses no sensitive variables, which is not the case. Take $\alpha = \frac{b}{c \cdot a}$.

If $N_{qr} = N'(V_1, \ldots, V_m)$ and $N_{db} = N'(W_1, \ldots, W_m)$ for some composite seminorms N', V_i, W_i, then it suffices to apply the aforementioned procedure only to such $i \in [m]$, where $V_i \geq W_i$. Let α_i be such, that $\alpha_i \cdot V_i \leq W_i$. By Lemma 3.75, we get $N'(\alpha_1 \cdot V_1, \ldots, \alpha_m \cdot V_m) \leq N'(W_1, \ldots, W_m)$. We can now take $\alpha = \min_i \alpha_i$.

3.3.8.5 Privacy vs Utility. The more noise is added to the output, the better privacy guarantees we get. However, the utility of the released differentially private output decreases. One needs to find the balance between privacy and utility. In general, utility is a quite application-specific measure, and one application may require better output accuracy than another one.

A possible generic solution of estimating utility is to give an upper bound on the noise magnitude. Since widely used additive noise distributions (Laplace, Cauchy, Gaussian) are unbounded, instead of fixing a strict upper bound, we can only define a range within which the noise stays with a sufficiently high confidence.

We note that there also exist DP mechanisms with nice bounded distributions like truncated Laplace [57]. However, they only provide (ϵ, δ)-DP for $\delta > 0$. Such mechanisms can be used in the cases where the error bound should hold for sure, but the privacy bound is more relaxed, and leakage is allowed with a reasonable probability that depends on δ.

Probabilistic upper bound on noise magnitude. The noise magnitude depends linearly on the quantity $\lambda := \frac{c(t)}{b}$, where $c(t)$ is the derivative sensitivity at point t (t is the actual data) and b a parameter that depends on ϵ and β, e.g. $b = (\frac{\epsilon}{\gamma} - \beta)$ for Cauchy distribution $GenCauchy(\gamma)$. The noise that is eventually added to a numeric output is $\lambda \cdot \eta$, where η is sampled from Cauchy distribution. The value λ itself does not give user any intuition how large that noise is. Instead, we would like to get an upper bound on the added noise. Unfortunately, both Laplace and Cauchy distributions are unbounded, and such an upper bound does not exist. However, we can still report a value λ such that error stays below λ with a certain probability (confidence). Knowing that $x \sim \frac{\sqrt{2}}{\pi} \cdot \frac{1}{1+|x|^4}$, we can compute

$$\int_{-1}^{1} \frac{\sqrt{2}}{\pi} \cdot \frac{1}{1+|x|^4} dx \approx 0.78 \ ,$$

thus fixing the probability that noise stays below $\lambda = \frac{c(t)}{b}$ to a constant $p = 0.78$. We would like to make our result more flexible and let the user choose p. Denoting PDF of noise distribution by $f_\eta(x)$, we need to find a such that

$$\int_{-a}^{a} f_\eta(x)\, dx = p \ .$$

For Laplace distribution, the equation $\int_{-a}^{a} \frac{\varepsilon}{2} \cdot e^{-|x|\varepsilon} dx = p$ reduces to $1 - e^{-a\varepsilon} = p$, from which we get $a = \frac{-\ln(1-p)}{\varepsilon}$. Unfortunately, there is no nice solution for Cauchy noise. However, since $\int_{-a}^{a} f_\eta(x)\, dx = 2 \cdot \int_{0}^{a} f_\eta(x)\, dx$, and $f_\eta(x)$ is monotone in $[0, \infty)$, we can use window binary search to find a for non-scaled Cauchy distribution. Now, in order to get probability p for $x = \lambda\eta$, we just need to take $a\lambda$ instead of a, as we have

$$p = \int_{-a}^{a} f_\eta(x)\, dx = \int_{-a}^{a} f_\eta(\lambda x)\, d(\lambda x) = \int_{-a\lambda}^{a\lambda} \frac{1}{\lambda} \cdot f_\eta(\lambda x)\, dx \ .$$

An upper bound on noise magnitude can be a fair estimate on its own, but it is not suitable for comparing different SQL queries. The badness of a depends on the query, its result, and its further use by the recipient of the query result. For example, the additive noise ±5 would almost have no effect on the actual count $y = 1000$, but it would be destructive for $y = 10$. For a single real output y, we can define *relative error* as $\frac{a}{|y|}$ (which is reasonable as far as $y \neq 0$). For a vector of outputs $\vec{y}$, we still want to have *one* number that characterizes the error. We propose to define the error as $\frac{\|\vec{a}\|_2}{\|\vec{y}\|_2}$, where $\vec{a}$ is the vector of errors.

Precision loss due to continuous approximation. We proposed to use sigmoids $\frac{e^{\alpha x}}{e^{\alpha x}+1}$ as continuous approximations of indicator functions $x > 0$ to implement filtering. Using sigmoids instead of the original filters causes a precision loss, as we use the value $\frac{e^{\alpha x}}{e^{\alpha x}+1}$ instead of 0 or 1. The difference is $\frac{1}{e^{\alpha|x|}+1}$, which goes to 0 when $|x| \to \infty$. The larger α is, the faster the difference goes to 0, and thus the smaller the precision loss. Unfortunately, we cannot increase α indefinitely to decrease the precision loss. If we want to achieve ϵ-differential privacy then the ϵ, together with the structure of the query, determines a value α_0 such that ϵ-differential privacy can only be achieved if $\alpha < \alpha_0$.

Suppose we have a SUM query using ℓ_1-norm to join row norms, and input rows are chosen from a certain distribution that does not change when input size (n, the number of rows) changes, which would

be a quite common scenario. Then the query result is roughly proportional in n, and the sensitivity (and thus the added noise) is roughly constant. Thus the relative error from added noise goes to 0 as $n \to \infty$. The relative error from sigmoids, however, is roughly constant as $n \to \infty$.

The larger n gets, the larger the error from sigmoids becomes relative to the error from added noise. We would like to decrease the former even if this increases the latter. We saw that we cannot increase α to achieve this. Instead, we use so-called *precise sigmoids*, keeping α constant but increasing the second parameter a as n increases.

Precise sigmoid. To get a higher precision than that of an ordinary sigmoid but still maintain α-smoothness, we use an extra parameter a in addition to α. We use the sigmoid $\sigma(x) = \frac{e^{ax}}{e^{ax}+1}$ but instead of its actual sensitivity $\sigma'(x)$, we use $c(x) = \frac{ae^{ax}}{(e^{ax}+1)^2}$, which is an α-smooth upper bound on $\sigma'(x)$. The smooth sensitivity is $\frac{a}{\alpha}$ times higher than that of the original sigmoid, but the difference from the precise filter value (0 or 1) is $\frac{e^{ax}+1}{e^{ax}+1}$ times smaller. If the probability density function of x is roughly constant near $x = 0$ then, for all $y \in (0, 1)$, the probability that the difference from the precise filter value is larger than y is

$$\Pr(\frac{1}{e^{ax}+1} > y) = \Pr(e^{ax} < \frac{1}{y} - 1) = \Pr(x < \frac{1}{a}\ln(\frac{1}{y} - 1)) \approx$$

$$\approx \frac{\alpha}{a}\Pr(x < \frac{1}{\alpha}\ln(\frac{1}{y} - 1)) = \frac{\alpha}{a}\Pr(\frac{1}{e^{\alpha x}+1} > y)$$

i.e. $\frac{a}{\alpha}$ times smaller than in the original sigmoid.

Then, assuming the probability density function of the input is roughly constant near the pivot point of the filter, the relative error from sigmoids will be roughly $\frac{a}{\alpha}$ times smaller and the relative error from added noise will be roughly $\frac{a}{\alpha}$ times larger.

Suppose that with an ordinary sigmoid the relative error from the sigmoid is k times larger than that from added noise. Then we can take $a = \alpha\sqrt{k}$ to make the two sources of error roughly equal and get the smallest amount of total noise. For the kind of queries considered above, k is roughly proportional to n, thus a would have to be roughly proportional to $\sqrt{n}$. The total error would then be inversely proportional to $\sqrt{n}$.

If more than one sigmoid is used then the total error would still be inversely proportional to $\sqrt{n}$. If no sigmoids are used then the total error would be inversely proportional to n.

We can get similar results for tauoids $\frac{2}{e^{-\alpha x}+e^{\alpha x}}$, used as continuous approximation of equality $x = 0$. Here we can allow nonzero probability at the pivot point, but near the pivot point (excluding the pivot point itself) the probability density function must still be roughly constant.

If the filtered values are integers then we do not have to use sigmoids and tauoids but instead use the precise functions $x > y \iff \min(1, \max(0, x - y))$ and $x = y \iff 1 - \min(1, \max(0, |x - y|))$. Then the only error is from adding noise and it would be inversely proportional to n.

Utility loss vs privacy loss. In the previous paragraphs, we considered how precision improves asymptotically when the number of rows $n \to \infty$. Now we consider how precision depends on the required privacy level. Let us define *utility loss* as the total relative error. Let us define *privacy loss* as the differential-privacy ϵ.

First consider the case without sigmoids. The relative error from noise is $E_n = \frac{c(x)}{bf(x)}$. Here c is a β-smooth upper bound on DS_f, thus it is monotonically decreasing in β. By default, we have $b = \beta = \frac{\epsilon}{2(\gamma+1)}$. Thus $E_n = \frac{2(\gamma+1)c(x)}{\epsilon f(x)}$ and c is also monotonically decreasing in ϵ. So if we increase ϵ by a factor of $k > 1$ then E_n decreases by a factor of at least k.

The global sensitivity can be defined as $\mathrm{GS}_f = \max_x \mathrm{DS}_f(x)$. Note that the product of utility loss and privacy loss, $E_n \cdot \epsilon$, is monotonically decreasing in ϵ and

$$\lim_{\epsilon\to 0} E_n \cdot \epsilon = \frac{2(\gamma+1)c(x)}{f(x)} \cdot \mathrm{GS}_f \ ,$$

$$\lim_{\epsilon\to \infty} E_n \cdot \epsilon = \frac{2(\gamma+1)c(x)}{f(x)} \cdot \mathrm{DS}_f(x) \ .$$

Thus $E_n \cdot \epsilon$ goes from $\frac{2(\gamma+1)c(x)}{f(x)} \cdot \mathrm{GS}_f$ to $\frac{2(\gamma+1)c(x)}{f(x)} \cdot \mathrm{DS}_f(x)$ as ϵ goes from 0 to ∞. Thus the larger ϵ is, the bigger the advantage of using smooth derivative sensitivity instead of global sensitivity.

Now consider the case that uses sigmoids. We optimize the tradeoff between sigmoid precision and added noise. The total relative error is $E = 2E_n$ and we get similar results for $E \cdot \epsilon$ as above for $E_n \cdot \epsilon$.

3.3.9 Combining Derivative Sensitivity for Row Multiplicities and Components. In Sec. 3.3.7, we showed how to compute smooth derivative sensitivity for relational algebra queries, where the derivatives are w.r.t. the multiplicities of rows in input tables. The distance between two databases is the number of rows we have to add to or remove from the first database to transform it into the second database. The shortcoming of this approach is that if the databases differ by only a small change in one row then the distance is the same as when they differ by a large change in that row (in both cases we need to remove one row and add another).

In Sec. 3.3.8, we showed how to compute smooth derivative sensitivity in Banach spaces, where the derivatives are Fréchet derivatives w.r.t. the values of rows in input tables. The distance between two databases quantifies the amount of changes we have to make to the values of the rows in the first database to transform it into the second database. This removes the shortcoming of the first approach but introduces another: the number of rows in the tables of the two databases must now be the same.

We would like to combine the two approaches, allowing to add and remove rows to/from the database and also make changes to individual rows so that small changes correspond to small distances.

Because in this section we will use two different kinds of derivative sensitivity (w.r.t. row multiplicities and w.r.t. components), let us call the derivative sensitivity w.r.t. components (i.e. in Banach spaces) *Banach sensitivity* or *Banach derivative sensitivity*. In ambiguous cases, we call the derivative sensitivity w.r.t. row multiplicities also *adding/removing rows sensitivity*.

3.3.9.1 Generalization of Derivative Sensitivity.

Extending the definitions. First, let us extend the notion of derivative sensitivity. We will need the following lemma.

Lemma 3.78. *Let X be a convex subset of a Banach space with norm $\|\ \|$, $f : X \to \mathbb{R}$ a function, $c : X \to \mathbb{R}_+$ a β-smooth upper bound on the derivative sensitivity (in the sense of Def. 3.55) of f. Then*

$$|f(x') - f(x)| \leq e^{\beta\|x'-x\|} c(x) \|x' - x\|$$

Proof. Let $x, x' \in X$. By the Mean Value Theorem for Banach spaces, there exists $\lambda \in (0, 1)$ such that

$$f(x') - f(x) = df_{(1-\lambda)x+\lambda x'}(x' - x)$$

Then

$$|f(x') - f(x)| \leq \|df_{(1-\lambda)x+\lambda x'}\| \cdot \|x' - x\| = \mathrm{DS}_f((1-\lambda)x + \lambda x') \cdot \|x' - x\|$$

Because c is an upper bound of DS_f,

$$\mathrm{DS}_f((1-\lambda)x + \lambda x') \leq c((1-\lambda)x + \lambda x')$$

Because c is β-smooth and $\|((1-\lambda)x + \lambda x') - x\| = \lambda\|x' - x\|$,

$$c((1-\lambda)x + \lambda x') \leq e^{\beta\lambda\|x'-x\|} c(x) \leq e^{\beta\|x'-x\|} c(x)$$

Thus

$$|f(x') - f(x)| \leq e^{\beta\|x'-x\|} c(x) \|x' - x\|$$

□

We have the following definition, which generalizes Definition 3.39 and Definition 3.55 to an arbitrary metric space instead of a Banach space:

Definition 3.58. Let (X, d) be a metric space. Let $f : X \to \mathbb{R}$. The *derivative sensitivity* of f is the following mapping from X to $\mathbb{R}_+$, where $\mathbb{R}_+$ denotes the set of all non-negative real numbers:

$$\mathrm{DS}_f(x) = \limsup_{x' \colon d(x,x') \to 0} \frac{|f(x) - f(x')|}{d(x, x')}$$

Note that because the argument of lim sup is a real-valued expression, the lim sup always exists (but may be infinite).

It is easy to see that if the derivative sensitivity in the sense of Def. 3.39 or Def. 3.55 exists then it is also the derivative sensitivity in the sense of Def. 3.58. The converse does not necessarily hold because Def. 3.39 and Def. 3.55 require the existence of an (ordinary or Fréchet) derivative but in Def. 3.58, the derivative sensitivity is always defined. This is similar to using the upper derivative instead of the ordinary derivative.

Lemma 3.79. *Let (X, d) be a metric space. Let $f : X \to \mathbb{R}$. If the derivative sensitivity of f is finite then f is continuous.*

Proof. Let $x \in X$.

$$\limsup_{x' \colon d(x,x') \to 0} |f(x) - f(x')| \leq \left(\limsup_{x' \colon d(x,x') \to 0} d(x, x') \right) \left(\limsup_{x' \colon d(x,x') \to 0} \frac{|f(x) - f(x')|}{d(x, x')} \right) = 0$$

Because $|f(x) - f(x')|$ is non-negative, it follows that

$$\lim_{x' \colon d(x,x') \to 0} |f(x) - f(x')| = 0$$

□

We also extend the definition of smoothness to the case where X is any metric space:

Definition 3.59. Let $p : X \to \mathbb{R}$ and $\beta \in \mathbb{R}$. The mapping p is *β-smooth*, if $p(x) \leq e^{\beta \cdot d(x,x)} \cdot p(x')$ for all $x, x' \in X$.

Combining the two analyses. Let

$$X = X_1 \times \cdots \times X_m$$

$$f : X \to \mathbb{R}$$

The filter predicate

$$F : X \to \mathbb{B}$$

The query

$$Q : (X_1 \to \mathbb{R}^+) \times \cdots \times (X_m \to \mathbb{R}^+) \to \mathbb{R}$$

Elements of $X_i \to \mathbb{R}^+$ are *generalized multisets* where multiplicities need not be integers.

$$Q(T_1, \ldots, T_m) = \sum \{\!\{ f(x) \mid x \in T_1 \times \cdots \times T_m, F(x) \}\!\}$$

Let

$$c_1 : (X_1 \to \mathbb{R}^+) \times \cdots \times (X_m \to \mathbb{R}^+) \to \mathbb{R}$$

be a β-smooth upper bound on the derivative sensitivity (in the sense of Def. 3.39 and thus also in the sense of Def. 3.55) of Q w.r.t. the (scaled by a factor of $G \in \mathbb{R}^+$) distance d_1:

$$d_1((T_1, T_2, \ldots, T_m), (T_1', T_2, \ldots, T_m)) = G \sum_{x \in X_1} |T_1(x) - T_1'(x)|$$

(i.e. $d_1(Y, Y')$ is G times the number of rows needed to add to or remove from the table T_1 of Y to obtain Y'). Note that X is a convex subset of a Banach space with the norm corresponding to d_1. Also note that we are only considering changes in the table T_1, the other tables are considered to be fixed. This is because we currently only know how to compute the combined sensitivity w.r.t. each table separately, and w.l.o.g. we can assume that the table w.r.t. which we are computing the sensitivity, is T_1.

Let X_1 be an open convex subset of a Banach space and d_0 the distance corresponding to the norm of that Banach space. We use d_0 to define a distance d_2 between some pairs of two databases Y and Y' that differ by at most one row in table T_1 (and are equal in all other tables):

$$d_2(Y, Y') = d_0(r, r')$$

where r is a row in table T_1 of Y and r' is a row in table T_1 of Y' such that removing r from Y and r' from Y' results in the same database, and r and r' may differ only in the columns considered sensitive in the analysis that uses d_2. For all other pairs of databases Y and Y', let $d_2(Y, Y') = \infty$. Thus d_2 is an extended distance. Note that the relation $\{(Y, Y') \mid d_2(Y, Y') < \infty\}$ is an equivalence relation whose equivalence classes are open convex subsets of Banach spaces with norm corresponding to the distance d_2.

Let

$$c_0 : (X_1 \to \mathbb{R}^+) \times \cdots \times (X_m \to \mathbb{R}^+) \to \mathbb{R}$$

be a β-smooth upper bound on the derivative sensitivity (in the sense of Def. 3.55) of Q w.r.t. the distance d_2.

We combine the two distances d_1 and d_2 using a form of edit distance. Suppose that we want to find the combined distance between databases Y and Y'. Then we may transform Y into Y' through intermediary databases $Y = Y_1, Y_2, \ldots, Y_{n-1}, Y_n = Y'$ such that $\sum_{i=1}^{n-1} d_{k_i}(Y_i, Y_{i+1})$ where $k_i \in \{1, 2\}$, is minimized over the choice of n, Y_i and k_i. This minimum (actually infimum, if $n \to \infty$) can be taken as the combined distance $d(Y, Y')$:

Definition 3.60. Let d_1 and d_2 be extended distances on the set X. The combined distance of d_1 and d_2 is d where for all $Y, Y' \in X$,

$$d(Y, Y') = \inf_{\substack{n \in \mathbb{N} \\ Y_1, \ldots, Y_n \in X \\ k_1, \ldots, k_n \in \{1,2\}}} \sum_{i=1}^{n-1} d_{k_i}(Y_i, Y_{i+1}) \tag{34}$$

(34) is equivalent to

$$\forall \epsilon > 0.\ \exists n, Y_1, \ldots, Y_n, k_1, \ldots, k_n.\ d(Y, Y') \leq \sum_{i=1}^{n-1} d_{k_i}(Y_i, Y_{i+1}) < d(Y, Y') + \epsilon \tag{35}$$

It is easy to see that the triangle inequality holds for d.

Lemma 3.80. *The combined distance d of two extended distances d_1 and d_2 is an extended distance. If one of d_1 and d_2 is a distance then also d is a distance.*

As d_1 is a distance and d_2 is an extended distance, their combined distance d is a distance.

We would like to combine the smooth sensitivities c_1 and c_0 to obtain a smooth sensitivity of Q w.r.t. the combined distance d. Intuitively, we would take $\max(c_1, c_0)$ as the combined sensitivity. This would work if c_1 and c_0 were smooth w.r.t. d but they are smooth only w.r.t. d_1 and d_2, respectively.

To make c_1 smooth w.r.t. d_2, we restrict the analysis that computes c_1 so that c_1 will not depend on the columns considered sensitive in the analysis that computes c_2. This may increase c_1 somewhat but it seems not much in practice.

To make c_0 smooth w.r.t. d_1, we also compute the global maximum of the sensitivity of f, which does not depend on the input database:

$$c_3 = \max_{x \in X} \mathrm{DS}_f(x)$$

and the sensitivity $c_4(Y)$ of the COUNT query corresponding to Q w.r.t. distance d_1. Then adding/removing/exchanging δ (an infinitesimal number of) rows in T_1 will add/remove/exchange at most $\delta c_4(Y)$ rows in the filtered cross product. Changing the value of those δ rows using distance d_2 will have sensitivity at most $\delta c_3 c_4(Y)$.

If Q uses T_1 only once then $c_4(Y)$ does not depend on the rows in T_1 and then adding/removing/exchanging 1 row in T_1 will add/remove/exchange at most $c_4(Y)$ rows in the filtered cross product. Changing the value of this 1 row using distance d_2 will have sensitivity at most $c_3 c_4(Y)$. As adding/removing/exchanging 1 row corresponds to d_1-distance G, the sensitivity of Q w.r.t. d_2 at d_1-distance G from Y is at most $c_3 c_4(Y)$. If this sensitivity is actually reached then the upper bound on the sensitivity of Q w.r.t. d_2 at Y must be at least $e^{-\beta G} c_3 c_4(Y))$ to maintain smoothness w.r.t. d_1.

This motivates us to take

$$c_2(Y) = \max(c_0(Y), e^{-\beta G} c_3 c_4(Y))$$

as the upper bound on the sensitivity of Q w.r.t. d_2. We want that c_2 would be smooth w.r.t. d_1.

Suppose that Q uses T_1 only once and row multiplicities are restricted to be integers. In this case, the d_1-distances are all multiples of G (thus cannot be between 0 and G) and sensitivity at any d_1-distance from Y is at most $c_3 c_4(Y)$. Note that c_4 here is technically local sensitivity but it is in this case equal to the derivative sensitivity. Thus c_2 is smooth w.r.t. d_1.

Suppose that Q uses T_1 more than once and row multiplicities are still restricted to be integers. In this case, the d_1-distances are still all multiples of G (thus cannot be between 0 and G) and sensitivity at any d_1-distance from Y is at most $c_3 c_4(Y)$ where now c_4 is the smooth *local* sensitivity of the COUNT query corresponding to Q w.r.t. distance d_1. Thus c_2 is smooth w.r.t. d_1.

Now, if c_2 is smooth w.r.t. d_1, we have that $c = \max(c_1, c_2)$ is β-smooth w.r.t. d_1 and also w.r.t. d_2.

Lemma 3.81. *Let $f : X \to \mathbb{R}$. Let d_1 and d_2 be extended distances on X and d their combined distance. Let f be β-smooth w.r.t. d_1 and also w.r.t. d_2. Then f is also β-smooth w.r.t. d.*

Proof. We prove the statement for distances. It is easy to generalize it to extended distances.

We prove by induction over n that if $Y = Y_1, Y_2, \ldots, Y_{n-1}, Y_n = Y'$ where $k_i \in \{1, 2\}$ and $d_{k_i}(Y_i, Y_{i+1}) < \infty$ then

$$f(Y') \leq e^{\beta \sum_{i=1}^{n-1} d_{k_i}(Y_i, Y_{i+1})} f(Y)$$

The proof is given by the following line:

$$f(Y_n) \leq e^{\beta d_{k_{n-1}}(Y_{n-1}, Y_n)} f(Y_{n-1}) \leq e^{\beta d_{k_{n-1}}(Y_{n-1}, Y_n)} e^{\beta \sum_{i=1}^{n-2} d_{k_i}(Y_i, Y_{i+1})} f(Y) = e^{\beta \sum_{i=1}^{n-1} d_{k_i}(Y_i, Y_{i+1})} f(Y)$$

where the first inequality follows from the smoothness of f w.r.t. d_{k-1} and the second from the induction hypothesis. The induction base ($n = 1$) is trivial.

Now we use (35) and get that for all $\epsilon > 0$, there exist $n, Y_1, \ldots, Y_n, k_1, \ldots, k_n$ such that

$$\sum_{i=1}^{n-1} d_{k_i}(Y_i, Y_{i+1}) < d(Y, Y') + \epsilon$$

Then

$$f(Y') < e^{\beta(d(Y,Y')+\epsilon)} f(Y)$$

This holds for all $\epsilon > 0$, thus

$$f(Y') \leq \inf_{\epsilon > 0} e^{\beta(d(Y,Y')+\epsilon)} f(Y) = e^{\beta d(Y,Y')} f(Y)$$

Similarly, we get

$$f(Y) \leq e^{\beta d(Y,Y')} f(Y')$$

Thus f is β-smooth w.r.t. d. □

Thus c is β-smooth w.r.t. d.

We know that c is an upper bound on the derivative sensitivity of Q w.r.t. d_1 and also w.r.t. d_2.

Lemma 3.82. *Let $Q : X \to \mathbb{R}$. Let d_1 and d_2 be extended distances on X such that for all $i \in \{1, 2\}$, $x \in X$, the set $\{x' \mid d_i(x, x') < \infty\}$ is a convex subset of a Banach space. Let d be the combined distance of d_1 and d_2. Let c be a β-smooth w.r.t. d upper bound on the derivative sensitivity of Q w.r.t. d_1 and also w.r.t. d_2. Then it is also an upper bound on the derivative sensitivity of Q w.r.t. d.*

Proof. Let Y be arbitrary. Let $\epsilon > 0$ be arbitrary. Let

$$\epsilon_1 = \min(1, \sqrt{1 + \frac{\epsilon}{c(Y)}} - 1)$$

$$\delta = \epsilon_2 = \frac{1}{8\beta} \ln\left(1 + \frac{\epsilon}{c(Y)}\right)$$

Then

$$\epsilon_1 \delta \leq \epsilon_2$$

Let Y' be such that $d(Y, Y') < \delta$. Now we use (35), taking $\epsilon = \epsilon_1 d(Y, Y')$, and get $n, Y_1, \ldots, Y_n, k_1, \ldots, k_n$ such that

$$\sum_{i=1}^{n-1} d_{k_i}(Y_i, Y_{i+1}) < d(Y, Y') + \epsilon_1 d(Y, Y')$$

Then also

$$\forall i.\ d_{k_i}(Y_i, Y_{i+1}) < d(Y, Y') + \epsilon_1 d(Y, Y') < \delta + \epsilon_1 \delta \leq \delta + \epsilon_2$$

and from (34),

$$\forall i.\ d(Y, Y_i) \leq \sum_{j=1}^{i-1} d_{k_j}(Y_j, Y_{j+1}) < d(Y, Y') + \epsilon_1 d(Y, Y') \leq \delta + \epsilon_2$$

Using Lemma 3.78 and that c_{k_i} is β-smooth over d_{k_i},

$$|Q(Y) - Q(Y')| \leq \sum_{i=1}^{n-1} |Q(Y_i) - Q(Y_{i+1})| \leq \sum_{i=1}^{n-1} e^{\beta d_{k_i}(Y_i, Y_{i+1})} c_{k_i}(Y_i) d_{k_i}(Y_i, Y_{i+1}) \leq e^{\beta(\delta+\epsilon_2)} \sum_{i=1}^{n-1} c_{k_i}(Y_i) d_{k_i}(Y_i, Y_{i+1})$$

Using that c_{k_i} is β-smooth over d,

$$c_{k_i}(Y_i) \leq e^{\beta d(Y, Y_i)} c_{k_i}(Y)$$

$$|Q(Y) - Q(Y')| \leq e^{\beta(\delta+\epsilon_2)} \sum_{i=1}^{n-1} e^{\beta d(Y, Y_i)} c_{k_i}(Y) d_{k_i}(Y_i, Y_{i+1}) \leq e^{2\beta(\delta+\epsilon_2)} \sum_{i=1}^{n-1} c_{k_i}(Y) d_{k_i}(Y_i, Y_{i+1})$$

$$|Q(Y) - Q(Y')| \leq e^{2\beta(\delta+\epsilon_2)} c(Y) \sum_{i=1}^{n-1} d_{k_i}(Y_i, Y_{i+1}) < e^{2\beta(\delta+\epsilon_2)} c(Y)(d(Y, Y') + \epsilon_1 d(Y, Y'))$$

$$|Q(Y) - Q(Y')| < e^{2\beta(\delta+\epsilon_2)}(1 + \epsilon_1) c(Y) d(Y, Y')$$

$$e^{2\beta(\delta+\epsilon_2)}(1 + \epsilon_1) \leq 1 + \frac{\epsilon}{c(Y)}$$

$$|Q(Y) - Q(Y')| < \left(1 + \frac{\epsilon}{c(Y)}\right) c(Y) d(Y, Y') = (c(Y) + \epsilon) d(Y, Y')$$

$$\frac{|Q(Y) - Q(Y')|}{d(Y, Y')} < c(Y) + \epsilon$$

We have proved that

$$\forall Y.\ \forall \epsilon > 0.\ \exists \delta > 0.\ \forall Y'.\ d(Y, Y') < \delta \Rightarrow \frac{|Q(Y) - Q(Y')|}{d(Y, Y')} < c(Y) + \epsilon$$

$$\forall Y.\ \limsup_{Y' :\, d(Y,Y')\to 0} \frac{|Q(Y) - Q(Y')|}{d(Y, Y')} \le c(Y)$$

Thus c is an upper bound on the derivative sensitivity of Q w.r.t. d. □

Thus c is an upper bound on the derivative sensitivity of Q w.r.t. d.

Combining Lemmas 3.81 and 3.82 easily gives us the following theorem:

Theorem 3.83. *Let $Q : X \to \mathbb{R}$. Let d_1 and d_2 be extended distances on X such that for all $i \in \{1, 2\}$, $x \in X$, the set $\{x' \mid d_i(x, x') < \infty\}$ is a convex subset of a Banach space. Let c be a β-smooth w.r.t. d_1 and also w.r.t. d_2, upper bound on the derivative sensitivity of Q w.r.t. d_1 and also w.r.t. d_2. Then c is also a β-smooth w.r.t. d upper bound on the derivative sensitivity of Q w.r.t. d.*

Thus c is a β-smooth upper bound on the derivative sensitivity of Q w.r.t. d.

3.3.9.2 From Combined Derivative Sensitivity to Differential Privacy. Let $Q : (X_1 \to \mathbb{R}^+) \times \cdots \times (X_m \to \mathbb{R}^+) \to \mathbb{R}$ be a query. Suppose that we have two sensitivities:

$$c_1 : (X_1 \to \mathbb{R}^+) \times \cdots \times (X_m \to \mathbb{R}^+) \to \mathbb{R}$$

is a β-smooth upper bound on the derivative sensitivity of Q w.r.t. the distance d_1, and

$$c_0 : (X_1 \to \mathbb{R}^+) \times \cdots \times (X_m \to \mathbb{R}^+) \to \mathbb{R}$$

is a β-smooth upper bound on the derivative sensitivity of Q w.r.t. the distance d_2.

We showed that if we have a function c that is smooth w.r.t. both initial distances and is an upper bound on the derivative sensitivity w.r.t. both initial distances, then it is also a smooth upper bound on the derivative sensitivity w.r.t. the combined distance.

To get differential privacy w.r.t. d_0 and c_0, we use Theorem 3.62. To extend the differential privacy to d_2 and c_0, we use the following lemma:

Lemma 3.84. *Let d be an extended distance on X. Suppose that for all $x \in X$, $g : X' \to \mathcal{D}(\mathbb{R})$ is ϵ-differentially private w.r.t. d restricted to the set X', where $X' = \{x' \mid d_i(x, x') < \infty\}$. Then g is also ϵ-differentially private w.r.t. d.*

Proof. Let $x, x' \in X$. We have to show that $d_{\mathrm{dp}}(g(x), g(x')) \le \epsilon \cdot d(x, x')$. If $d(x, x') < \infty$ then $x' \in \{x' \mid d_i(x, x') < \infty\}$ and thus $d_{\mathrm{dp}}(g(x), g(x')) \le \epsilon \cdot d(x, x')$. If $d(x, x') = \infty$ then $d_{\mathrm{dp}}(g(x), g(x')) \le \infty = \epsilon \cdot d(x, x')$. □

To get differential privacy w.r.t. d_1 and c_1, we use the Theorem 3.53. If instead of derivative adding/removing rows sensitivity, we had a smooth upper bound on the *local* adding/removing rows sensitivity then we could use (but currently are not using) the following lemma instead of Theorem 3.53:

Lemma 3.85. *Let $f : X \to \mathbb{R}$. Let d be an integer-valued distance on X such that for all $x, x' \in X$, there exist $x_0, \ldots, x_n \in X$ where $n = d(x, x')$, such that $x_0 = x, x_n = x'$ and for $i = 0, \ldots, n-1$, $d(x_i, x_{i+1}) = 1$. Let $\gamma, b, \beta \in \mathbb{R}_+$, $\gamma > 1$. Let $\epsilon = (\gamma + 1)(b + \beta)$. Let η be a random variable distributed according to GenCauchy(γ). Let c be a β-smooth upper bound on the local sensitivity of f. Then $g(x) : f(x) + \frac{c(x)}{b} \cdot \eta$ is ϵ-differentially private.*

Proof. Let $x, x' \in X$. Then there exist $x_0, \ldots, x_n \in X$ where $n = d(x, x')$, such that $x_0 = x, x_n = x'$ and for $i = 0, \ldots, n-1$, $d(x_i, x_{i+1}) = 1$.

$$d_{\mathrm{dp}}(g(x), g(x')) \le \sum_{i=0}^{n-1} d_{\mathrm{dp}}(g(x_i), g(x_{i+1})) \le (\gamma + 1) \cdot \left(b \cdot \frac{|f(x_{i+1}) - f(x_i)|}{c(x)} + \beta d(x_i, x_{i+1}) \right)$$

$$d_{\mathrm{dp}}(g(x), g(x')) \le \sum_{i=0}^{n-1} (\gamma + 1) \cdot (b \cdot d(x_i, x_{i+1}) + \beta \cdot d(x_i, x_{i+1}))$$

$$d_{\mathrm{dp}}(g(x), g(x')) \leq \sum_{i=0}^{n-1} (\gamma+1)(b+\beta) \cdot d(x_i, x_{i+1}) = (\gamma+1)(b+\beta) \cdot n = (\gamma+1)(b+\beta) \cdot d(x, x') \ .$$

□

Now we prove a theorem that allows combining differential privacy w.r.t. two different distances.

Theorem 3.86. *Let d_1 and d_2 be extended distances on X. Let d be the combined distance of d_1 and d_2. Suppose that $g : X \to \mathcal{D}(\mathbb{R})$ is ϵ-differentially private w.r.t. d_1 and also w.r.t. d_2. Then g is also ϵ-differentially private w.r.t. d.*

Proof. Let $x, x' \in X$. We have to show that $d_{\mathrm{dp}}(g(x), g(x')) \leq \epsilon \cdot d(x, x')$.

Let $\epsilon > 0$ be arbitrary. From (35),

$$\exists n, x_1, \ldots, x_n, k_1, \ldots, k_n. \ \sum_{i=1}^{n-1} d_{k_i}(x_i, x_{i+1}) < d(x, x') + \epsilon \ .$$

Because g is ϵ-differentially private w.r.t. d_{k_i},

$$d_{\mathrm{dp}}(g(x_i), g(x_{i+1}) \leq \epsilon \cdot d_{k_i}(x_i, x_{i+1}) \ .$$

By the triangle inequality for d_{dp},

$$d_{\mathrm{dp}}(g(x), g(x')) \leq \sum_{i=1}^{n-1} d_{\mathrm{dp}}(g(x_i), g(x_{i+1})) \leq \epsilon \cdot \sum_{i=1}^{n-1} d_{k_i}(x_i, x_{i+1})$$

$$d_{\mathrm{dp}}(g(x), g(x')) < \epsilon(d(x, x') + \epsilon) \ .$$

Because this holds for all $\epsilon > 0$, we have

$$d_{\mathrm{dp}}(g(x), g(x')) \leq \epsilon \cdot d(x, x') \ .$$

□

To use Theorem 3.86, we need a single g to be differentially private w.r.t. both d_1 and d_2. The g is determined by the c, and so far we have differential privacy w.r.t. d_1 for c_1 and w.r.t. d_2 for c_0. We need differential privacy w.r.t. d_1 and d_2 for a single c. The theorems we used so far would still hold if c_1 and c_0 are replaced by a β-smooth upper bound on both. We will show how to find such a smooth upper bound in Sec. 3.3.9.3.

3.3.9.3 Smoothing. To use the theorems of Sec. 3.3.9.2 for obtaining differential privacy, we need a function c that is smooth w.r.t. both initial distances and is an upper bound on the derivative sensitivity w.r.t. both initial distances.

We already have c_1 smooth w.r.t. d_1 and c_0 smooth w.r.t. d_2. We need to cross-smoothe c_1 w.r.t. d_2 and c_0 w.r.t. d_1, i.e. make sensitivity w.r.t. each initial distance smooth w.r.t. the other initial distance.

First, we consider how to make sensitivity w.r.t. one initial distance in one table smooth w.r.t. the other initial distance in the same table.

Let f be the function that corresponds to the query Q. In Sec. 3.3.9.1, we proposed taking

$$c_2(Y) = \max(c_0(Y), e^{-\beta G} c_3 c_4(Y))$$

as the upper bound on the sensitivity of Q w.r.t. d_2, where $c_3 = \max_{x \in X} \mathrm{DS}_f(x)$ and $c_4(Y)$ is the sensitivity of the `COUNT` query corresponding to Q w.r.t. distance d_1.

We have made c_2 smooth w.r.t. d_1 but c_2 is defined only for databases Y with integer row multiplicities, unlike c_1, which was defined for databases Y with real-valued row multiplicities. This is not a problem since the actual databases used in practice have only integer row multiplicities and we

need differential privacy only for those. The fractional multiplicities are only used to make c_1 easier to compute.

We still have another problem. With $c = \max(c_1, c_2)$ we have c (unlike c_1) defined only for integer row multiplicities and we cannot use Theorem 3.53 which requires it to be defined for real-valued row multiplicities. We use the following modification of Theorem 3.53 to get differential privacy for c and d_1:

Theorem 3.87. *Let $X = (\mathbb{R}^n, d)$ be a metric space, with d being the ℓ_1-distance. Let*

$$\mathrm{DS}_f(\vec{x}) = \max_i \left| \frac{\partial f}{\partial x_i}(\vec{x}) \right| \ .$$

Let $\gamma, b, \beta \in \mathbb{R}_+$, $\gamma > 1$. Let $\epsilon = (\gamma + 1)(b + \beta)$. Let η be a random variable distributed according to GenCauchy(γ). Let $c = \max(c_1, c_2)$ where c_1 is a β-smooth upper bound on DS_f for a function $f : X \to \mathbb{R}$ and c_2 is β-smooth for inputs with integer row multiplicities. Then $g(\vec{x}) : f(\vec{x}) + \frac{c(\vec{x})}{b} \cdot \eta$ is ϵ-differentially private for inputs with integer row multiplicities (a finite number of which are nonzero).

Proof. Let $\vec{x}, \vec{x}' \in X$ and let all their coordinates be integers. Suppose that they differ only in the i_0-th coordinate. W.l.o.g. assume that $x'_{i_0} \geq x_{i_0}$. Denote $L = x'_{i_0} - x_{i_0}$. We have to show that $d_{\mathrm{dp}}(g(\vec{x}'), g(\vec{x})) \leq \epsilon L = (\gamma + 1)(b + \beta)L$.

We extend c_2 to inputs on the segment connecting $\vec{x}$ and $\vec{x}'$, where the i_0-th coordinate can be real-valued (the rest must still be integer-valued, we do not know whether such extension is possible for more than one coordinate at a time).

$$c_2(\vec{x}[n + r]) := c_2(\vec{x}[n])^{1-r} \cdot c_2(\vec{x}[n + 1])^r \quad \text{for } n \in \mathbb{N}, r \in (0, 1)$$

It is easy to see (using the smoothness of c_2 for integer multiplicities) that this extended c_2 is β-smooth on the segment connecting $\vec{x}$ and $\vec{x}'$. Then also $c = \max(c_1, c_2)$ using this extended c_2 is a β-smooth upper bound on DS_f on the segment connecting $\vec{x}$ and $\vec{x}'$.

Now we can use Theorem 3.53 to get $d_{\mathrm{dp}}(g(\vec{x}'), g(\vec{x})) \leq (\gamma + 1)(b + \beta)L$ since its proof only uses c on the segment connecting $\vec{x}$ and $\vec{x}'$.

If $\vec{x}$ and $\vec{x}'$ differ in more than one coordinate, then we can transform $\vec{x}$ to $\vec{x}'$ by changing one coordinate at a time, and using the triangle inequality. □

Getting differential privacy for c and d_2 is easier. As noted in Sec. 3.3.9.1, to make c_1 smooth w.r.t. d_2, we restrict the analysis that computes c_1 so that c_1 will not depend on the columns considered sensitive in the analysis that computes c_2. The same restriction makes c_4 smooth w.r.t. d_2. Also c_0 is smooth w.r.t. d_2. Thus also c will be smooth w.r.t. d_2 and we can use Theorem 3.62 and Lemma 3.84 to obtain differential privacy for c and d_2.

Now we use Theorem 3.86 to get differential privacy for c and d. Note that, even though we use fractional row multiplicities in some parts of the analysis to make computations easier, the differential privacy we obtain in the end is only proved for integer multiplicities, i.e. the query is restricted to

$$Q : (X_1 \to \mathbb{N}) \times \cdots \times (X_m \to \mathbb{N}) \to \mathbb{R} \ .$$

As we noted before, this is sufficient in practice.

Allowing G to depend on the table. Suppose that instead of

$$d_1((T_1, T_2, \ldots, T_m), (T'_1, T'_2, \ldots, T'_m)) = G \sum_{i=1}^{m} \sum_{x \in X_i} |T_i(x) - T'_i(x)|$$

we have

$$d_1((T_1, T_2, \ldots, T_m), (T'_1, T'_2, \ldots, T'_m)) = \sum_{i=1}^{m} G_i \sum_{x \in X_i} |T_i(x) - T'_i(x)|$$

where G_i is the distance between two databases that differ by one row in table T_i and are equal in the other tables.

If some of the tables are used more than once in the query then, because we first compute sensitivity separately w.r.t. each copy of a multiply-used table, we must divide G_i by the number of times the table T_i is used, so that when we add one row to each copy of T_i the total distance is still G_i.

Combined sensitivity w.r.t. one table. Derivative sensitivity can be smoothed exactly in the same way as it was done in the paragraph "Smoothing the derivative sensitivity" of Sec. 3.3.7.3. So far, we discovered that we also need smooth local sensitivity (c_4) in addition to smooth derivative sensitivity (c_1). An upper bound on the local sensitivity can be found by taking the supremum of the derivative sensitivities at distances in the range $[0, 1]$ because we can add/remove one row in infinitesimal increments and during this process, the derivative sensitivity never exceeds this upper bound.

As we noted before, the derivative sensitivity is monotonically increasing in the row multiplicities, thus we can find this upper bound by just considering adding 1 row to the database, thus taking the derivative sensitivity at distance 1.

Because we are computing the smooth sensitivity w.r.t. (one copy of) one table at a time, we add the 1 row to each of the copies of T_i (the table w.r.t. which we are currently computing the sensitivity). When computing $A(k)$, the sensitivity at distance k defined in (33), this added 1 row does not count towards the distance budget k. Thus

$$A(k) = \mathsf{Count}_{\sigma_\theta(\prod_{j\neq i} T_j)}(D) + \prod_{j\neq i}(n_j + \ell_j + k_j) - \prod_{j\neq i} n_j$$

where ℓ_j is 1 if T_j is a copy of the table T_i, and 0 otherwise. Then the derivative of $A(k)$ is

$$\frac{\partial(A(k))}{\partial k} = \frac{1}{G_m(n_m + \ell_m + k_m)} \prod_{j\neq i}(n_j + \ell_j + k_j)$$

and

$$\frac{\partial L}{\partial k} = -\beta + \frac{1}{G_m(n_m + \ell_m + k_m)} \cdot \left(1 + \frac{C}{\prod_{j\neq i}(n_j + \ell_j + k_j) - C}\right) .$$

The rest of the argument is similar to the paragraph "Smoothing the derivative sensitivity" of Sec. 3.3.7.3.

Combined Sensitivity w.r.t. Multiple Tables. Let us consider how to make sensitivity w.r.t. one initial distance in one table smooth w.r.t. the other initial distance in other tables. First, let us assume that each table can be used at most once in a query.

With only one table, Banach sensitivity c_0 is smoothed w.r.t. adding/removing rows as

$$c_2(Y) = \max(c_0(Y), e^{-\beta G} c_3 c_4(Y)) .$$

With multiple tables, the Banach sensitivity c_{0i} w.r.t. the i^{th} table is smoothed w.r.t. adding/removing rows in the i^{th} table as

$$c_{2i}(Y) = \max(c_{0i}(Y), c_{3i} \max_j(e^{-\beta G_j} c_{4j}(Y))) .$$

The Banach sensitivity c_{0i} w.r.t. the i^{th} table is smoothed w.r.t. adding/removing rows in all tables as

$$c_{2i}(Y) = \max(c_{0i}(Y), e^{-\beta G_i} c_{3i} c_{4i}(Y), \max_{j\neq i}(e^{-\beta G_j}(c_{0i}(Y) + c_{3i} c_{4j}(Y)))) .$$

Adding one row to table j adds at most $c_{4j}(Y)$ rows to the joined table. Each of those rows may be affected by the same row r in table i. The total Banach sensitivity of those added rows is at most $c_{3i}c_{4j}(Y)$. In addition, the joined table may already contain some rows affected by r, with total Banach sensitivity at most $c_{0i}(Y)$. Thus the total Banach sensitivity of rows affected by r, after adding one row to table j, is at most $c_{0i}(Y) + c_{3i}c_{4j}(Y)$. Because this sensitivity is at distance G_j, we have to multiply it

by $e^{-\beta G_j}$. If $i = j$ then the term $c_{0i}(Y)$ is not needed because then the new rows are affected by the row added to table i and thus cannot be affected by an existing row of table i.

If $G_i = \infty$ then $c_{3i} = 0$,

$$e^{-\beta G_j}(c_{0i}(Y) + c_{3i}c_{4j}(Y)) \leq c_{0i}(Y) \ ,$$

and as $c_{0i}(Y)$ is already included in $c(Y)$, we can ignore the term $e^{-\beta G_j}(c_{0i}(Y) + c_{3i}c_{4j}(Y))$. If $G_j = \infty$ then we can also ignore that term, as it is zero.

Combined sensitivity w.r.t. tables used more than once. So far, each table was assumed to be used at most once. We will now lift this restriction.

Let $\mathcal{I}_I$ be the set of indices of table-copies corresponding to table I. The Banach sensitivity w.r.t. table I is at most the sum of the sensitivities w.r.t. each copy of I:

$$C_{0I} = \sum_{i \in \mathcal{I}_I} c_{0i} \ .$$

If table I is used only once ($\mathcal{I}_I = \{i\}$) then

$$C_{2I}(Y) = \max(c_{0i}(Y), e^{-\beta G_i} c_{3i} c_{4i}(Y), \max_{J \neq I}(e^{-\beta G_J}(c_{0i}(Y) + c_{3i} C_{4J}(Y))))$$

Adding one row to table J adds at most $C_{4J}(Y) = \sum_{j \in \mathcal{I}_J} c_{4j}(Y)$ rows to the joined table.

If table I is used more than once then

$$\begin{aligned} C_{2I}(Y) = \max(&C_{0I}(Y), \\ &e^{-\beta G_I} \max(C_{3I} C_{4I}(Y), C_{0I}(Y) + (C_{3I} - \min_{i \in \mathcal{I}_I} c_{3i}) C_{4I}(Y)), \\ &\max_{J \neq I}(e^{-\beta G_J}(C_{0I}(Y) + C_{3I} C_{4J}(Y)))) \ . \end{aligned}$$

Adding one row to table J adds at most $C_{4J}(Y) = \sum_{j \in \mathcal{I}_J} c_{4j}(Y)$ rows to the joined table. Each of those rows may be affected by the same row r in table I. The total Banach sensitivity of those added rows is at most $C_{3I}C_{4J}(Y)$. In addition, the joined table may already contain some rows affected by r, with total Banach sensitivity at most $C_{0I}(Y)$. Thus the total Banach sensitivity of rows affected by r, after adding one row to table J, is at most $C_{0I}(Y) + C_{3I}C_{4J}(Y)$. Because this sensitivity is at distance G_J, we have to multiply it by $e^{-\beta G_J}$.

If $I = J$ then we have two cases. If r is the added row then the term $C_{0I}(Y)$ is not needed because r cannot be an existing row of table i (multiple copies of the same row are counted as separate rows). Thus we get the term $C_{3I}C_{4I}(Y)$. If r is an existing row then it can affect each new row in the joined table through at most $|\mathcal{I}_I| - 1$ copies of table I, as at most one copy must provide the row newly added to table I. Thus the sensitivity of each new row in the joined table w.r.t. row r is at most $C_{3I} - \min_{i \in \mathcal{I}_I} c_{3i}$. Thus we get the term $C_{0I}(Y) + (C_{3I} - \min_{i \in \mathcal{I}_I} c_{3i})C_{4I}(Y)$.

3.3.9.4 Asymptotic Behavior.

Noise level. Suppose we have a sequence of databases $Y_1, Y_2, \ldots$ such that the number of rows in Y_n goes to infinity if $n \to \infty$. The result of a query Q on Y_n is $Q(Y_n)$. The magnitude of the added noise is proportional to $\frac{c(Y_n)}{b}$ where $c(Y_n)$ is a β-smooth upper bound on the combined sensitivity where $\varepsilon = (\gamma + 1)(b + \beta)$ and $\gamma > 1$ is a constant (by default $\gamma = 4$). If also ε and β are constant (independent of n) then b is constant and the magnitude of the added noise is proportional to $c(Y_n)$. The magnitude of the relative error is then proportional to $\frac{c(Y_n)}{Q(Y_n)}$.

We would like $\lim_{n \to \infty} \frac{c(Y_n)}{Q(Y_n)} = 0$. One way to achieve this is when $c(Y_n)$ is bounded and $Q(Y_n) \to \infty$.

We are considering SUM queries of the form

$$Q(Y) = \sum_{\substack{r \in \prod Y \\ \sigma(r)}} f(r)$$

where $\prod Y$ is the cross product of the tables in Y, σ is a predicate on rows, and f is a function from rows to real numbers. In the sequence $Y_1, Y_2, \ldots$, the number of rows in the database goes to infinity. Because the number of tables in the database is constant, there is at least one table t in which the number of rows goes to infinity. If there is at least one row in each of the other tables then the number of rows in $\prod Y$ goes to infinity. Suppose that $f(r) \geq 0$ for all r. If $f(r) = 0$ then we may take $\sigma(r) = 0$ thus we may assume $f(r) > 0$ for all r such that $\sigma(r) = 1$.

Suppose that we add rows to the table t and the rows are chosen from a probability distribution such the probability that the chosen row can be combined with one row from each of the other tables to a tuple r such that $\sigma(r) = 1$, is positive. Then, as the number of added rows goes to infinity, also the number of rows in $\prod Y$ that satisfy σ, goes to infinity. If there exist $\epsilon, \delta > 0$ such that $\Pr[f(r) \geq \delta] \geq \epsilon$, then also $Q(Y_n)$ goes to infinity.

Now consider $c(Y_n)$. We have

$$c(Y) = \max_i \max(c_{1i}(Y), c_{2i}(Y))) \ ,$$

$$c_{2i}(Y) = \max(c_{0i}(Y), e^{-\beta G_i} c_{3i} c_{4i}(Y), \max_{j \neq i}(e^{-\beta G_j}(c_{0i}(Y) + c_{3i} c_{4j}(Y)))) \ .$$

Because $c_{0i}(Y) \leq c_{3i} c_{4i}(Y)$, $e^{-\beta G_i} c_{3i} c_{4i}(Y) \leq c_{3i} c_{4i}(Y)$, and $e^{-\beta G_j}(c_{0i}(Y) + c_{3i} c_{4j}(Y)) \leq 2 c_{3i} c_{4i}(Y)$, we have

$$c_{2i}(Y) \leq 2 c_{3i} c_{4i}(Y) \ .$$

Note also that

$$c_{1i}(Y) \leq S \cdot c_{4i}(Y)$$

where S is the maximum possible value of $f(r)$ over all possible r. Thus

$$c(Y) \leq \max(S, 2 \max_i c_{3i}) \cdot \max_i c_{4i}(Y) \ .$$

Suppose that we add an infinite number of rows only to table i and the database contains more than one table. Then there is a row in table $j \neq i$ that is joined with an infinite number of rows from table i. Thus $c_{4j}(Y_n) \to \infty$. Thus we must require that an infinite number of rows be added to each of the private tables. Those tables that contain only a limited number of rows would usually be public anyway. If the number of rows with which a row of a private table is joined, is bounded then $c_{4j}(Y_n)$ (and thus also $c(Y_n)$) is bounded.

Suppose that there are two private tables and the number of rows in each goes to infinity at a different speed. If the number of rows r with non-infinitesimal value of $f(r)$ in the joined table grows faster than the number of rows in the larger input table then $\lim_{n\to\infty} \frac{c(Y_n)}{Q(Y_n)} = 0$ because $c(Y_n)$ can only grow as fast as the larger input table. If the maximum number of rows with which a row of an input table is joined (and thus $c(Y_n)$), grows slower than the number of rows r with non-infinitesimal value of $f(r)$ in the joined table then we also get $\lim_{n\to\infty} \frac{c(Y_n)}{Q(Y_n)} = 0$.

Suppose that there are n private tables and the number of rows in each goes to infinity at a different speed. If the number of rows r with non-infinitesimal value of $f(r)$ in the joined table grows faster than the maximum number of rows in a join of $n-1$ input tables (maximized over the choice of the $n-1$ tables out of n) then $\lim_{n\to\infty} \frac{c(Y_n)}{Q(Y_n)} = 0$ because $c(Y_n)$ can only grow as fast as a join of $n-1$ input tables.

Suppose that there are two private tables and the number of rows in each goes to infinity as $\Theta(n)$. Suppose that the two tables are joined by their primary keys. Suppose that the primary key columns are considered private in the Banach analyzer. Then the number of rows in the joined table will be $\Theta(n^2)$ but only $\Theta(n)$ of them will pass the filter. The maximum number of rows with which each row is joined is $\Theta(n)$ but only one of these rows will pass the filter. We do not get $\lim_{n\to\infty} \frac{c(Y_n)}{Q(Y_n)} = 0$ because both $c(Y_n) = \Theta(n)$ and $Q(Y_n) = \Theta(n)$. Suppose that we instead consider the primary key columns public in the Banach analyzer and handle their privacy in the local sensitivity analyzer (i.e. changing the primary key of a row is considered as exchanging the whole row). Unlike the Banach analyzer, the local sensitivity analyzer can apply the filters and thus there will be only $\Theta(n)$ rows in the joined tables and the maximum number of rows with which each row is joined, will be 1. Thus we will get $\lim_{n\to\infty} \frac{c(Y_n)}{Q(Y_n)} = 0$.

Local vs global Banach sensitivity. Consider the smoothing of the Banach sensitivity:

$$c_2(Y) = \max(c_0(Y), e^{-\beta G} c_3 c_4(Y)) \ .$$

Here $c_3c_4(Y)$ is the global Banach sensitivity. We mean global for the changes considered by the Banach analyzer. It may still change when rows are added or removed. Note that $c_2(Y)$ can be better than the global Banach sensitivity only by a factor of $e^{-\beta G}$. If we consider adding or removing one row to be one unit of change then $G = 1$. Also $\beta < \frac{\varepsilon}{5}$ where ε corresponds to this unit change. Thus the local Banach sensitivity cannot be better than the global by more than a factor of $e^{-\frac{\varepsilon}{5}}$. In practice, usually $\varepsilon \leq 1$ thus the factor will be less than $e^{0.2} \approx 1.2214$.

Thus local Banach sensitivity will have significant advantage over global Banach sensitivity only if the unit change is not adding or removing a row but a much smaller change inside a row. In this case, the attacker's advantage in guessing whether a certain row is included in the input is much larger than guessing whether a certain small change is applied to a certain input row. The term $e^{-\beta G} c_3 c_4(Y)$ is so large because $c_0(Y)$ can be increased to $c_3c_4(Y)$ by adding just one row to Y.

3.3.10 Computing Attacker Advantage in Guessing Some Value. In Sections 3.3.7, 3.3.8, 3.3.9, we showed how to achieve ϵ- (and (ϵ, δ)-) differential privacy (DP). The question is what is the appropriate value of ϵ, since there is no common agreement on a "sufficiently small" ϵ, and its goodness depends on the query as well as the data. In this section, we show how to compute ϵ that corresponds to ϵ', which is defined as the adversary's advantage in probability of guessing some specific property of the output.

We are using the attacker model of [58], where the attacker advantage is defined as the difference between its prior and posterior beliefs on the property that he is guessing. While [58] is based on Laplace noise distribution, the main DP mechanism of Section 3.3.8 is based on generalized Cauchy distribution, for which it is more difficult to compute precise bounds. Hence, we derive more general bounds, which can be derived directly from the definition of differential privacy, without relying on the specific privacy mechanism. We use some more general ideas from [59] that relate differential privacy to guessing probability.

Let $x \in X$ be the input. The attacker has a goal $g : X \to \{0,1\}^*$, which defines the information that he *wants to learn* about x. There is a function $e : X \to \{0,1\}^*$ defining the information that the attacker *already knows* about x. We want to know how the distribution of $g(X)$ changes after the attacker in addition gets the output $\mathcal{M}_f(x)$ of a (differentially private) query $\mathcal{M}_f : X \to Y$.

The security definition is related to the difference between posterior (with observation $\mathcal{M}_f(x)$) and prior (without observation $\mathcal{M}_f(x)$) probabilities of guessing $g(X) = g(x)$, assuming that the extra information $e(x)$ is included as a condition of both prior and posterior probabilities.

Definition 3.61. [Guessing advantage (discrete)] Let $x \in X$ be the data instance, $\mathcal{M}_f(x)$ the observation, g the attacker goal, and e the extra information. We say that advantage of guessing $g(x)$ is at most ϵ' if, for any algorithm A,

$$\Big|\Pr[A(\mathcal{M}_f(x), e(x)) = g(x) \mid Y] - \Pr[A(e(x)) = g(x) \mid Y]\Big| \leq \epsilon'$$

where $Y = (\mathcal{M}_f(X) = \mathcal{M}_f(x)) \wedge (e(X) = e(x))$.

For continuous data, the probability mass of each single point may be 0. It is possible that $\Pr[g(X) = g(x)] = 0$ for all x, so Def. 3.61 does not make any sense. Indeed, in practice it can be equally bad if the attacker learns some geographical location "precisely" or "close enough", so we need to introduce the notion of precision. The attacker wants to come up with a point x' such that $d(x, x') \leq r$ for a sufficiently small r.

Definition 3.62. [Guessing advantage (continuous)] Let $x \in X$ be the data instance, $\mathcal{M}_f(x)$ the observation, g the attacker goal, and e the extra information. Let $\mathcal{B}(x, r) = \{x' | d(x, x') \leq r\}$. We say that advantage of guessing $g(x)$ with precision r is at most ϵ' if, for any algorithm A,

$$\Big|\Pr[A(\mathcal{M}_f(x), e(x)) \in \mathcal{B}(g(x), r) \mid Y] - \Pr[A(e(x)) \in \mathcal{B}(g(x), r) \mid Y]\Big| \leq \epsilon'$$

where $Y = (\mathcal{M}_f(X) = \mathcal{M}_f(x)) \wedge (e(X) = e(x))$.

Depending on $y := \mathcal{M}_f(x)$, the attacker either wins or loses, so these definitions assume a distribution over y. For example, a mechanism that flips the input bit $x \in \{0, 1\}$ with probability $\frac{1}{1+e^{-\epsilon/2}}$ (i.e. exponential mechanism) satisfies ϵ-DP, but the attacker makes a correct guess in the cases where "the bit is not flipped" since the bit is always "less likely flipped than not". For this particular mechanism we know that the probability of flipping is exactly $\frac{1}{1+e^{-\epsilon/2}}$, and hence guessing advantage is $\frac{1}{1+e^{-\epsilon/2}} - \frac{1}{2} = \frac{1-e^{-\epsilon/2}}{2\cdot(1+e^{-\epsilon/2})}$. However, we cannot compute it from a general definition of DP mechanism, since the definition of DP universally quantifies over all y and does not distinguish between "good" and 'bad" y. Another approach is to estimate attacker's confidence in his guess, as it was done in [58], which can be done for an arbitrary y and is more related to DP guarantees.

We consider an attacker who knows the distribution of inputs, i.e. attacker's prior belief is equivalent to the actual distribution of inputs. If he only gets the extra knowledge $e(x)$, he estimates the probability weight of "sufficiently correct answers" $\mathcal{B}(g(x), r)$ as $\Pr_{pre}[\mathcal{B}(g(x), r)] := \Pr[X \in \mathcal{B}(g(x), r) \mid e(X) = e(x)]$, i.e. the *prior* probability. If he gets in addition the output $\mathcal{M}_f(x)$, the confidence is $\Pr_{post}[\mathcal{B}(g(x), r)] := \Pr_{pre}[\mathcal{B}(g(x), r) \mid \mathcal{M}_f(X) = \mathcal{M}_f(x), e(X) = e(x)]$, i.e. the *posterior* probability.

As in Section 3.3.8, we assume that a database is an element of a Banach space $X = (X_1, \ldots, X_n)$. In general, n is the total number of variables defining the database. While it seems the most intuitive to think that each X_i corresponds to some cell of a table in the database, it is possible that X_i in turn denotes a *subspace* of variables, i.e. a row or a column.

Let $f : X \to Y$ be a query. Let the noised query $\mathcal{M}_f$ be differentially private variant of f. Let $X = (X_1, \ldots, X_n)$, and let X_i be the component that the attacker wants to guess (e.g. a subset of table columns). Let us constrain Def. 3.62 and take $g(x) = x_k$ and $e(x) = (x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n)$.

Definition 3.63. [Guessing advantage w.r.t. component] Let $x = (x_1, \ldots, x_n) \in (X_1, \ldots, X_n)$ be the data instance, $\mathcal{M}_f(x)$ the observation. We say that advantage of guessing x_k with precision r_k is at most ϵ' if

$$\left| \Pr[\mathcal{B}(x_k, r_k) \mid \mathcal{M}_f(X) = \mathcal{M}_f(x), \bigwedge_{i \neq k} (X_i = x_i)] - \Pr[\mathcal{B}(x_k, r_k) \mid \bigwedge_{i \neq k} (X_i = x_i)] \right| \leq \epsilon' \ ,$$

where $\mathcal{B}(x_k, r_k) = \{x'_k | d(x_k, x'_k) \leq r\}$.

Formally, there is a collection $\mathcal{S} = \{S_k \mid k \in [|\mathcal{S}|]\}$ of sets of sensitive attributes that the attacker wants to guess. Each $S_k \in \mathcal{S}$ is of the form $S_k = \{(x_{k_1}, r_{k_1}), \ldots, (x_{k_{n_k}}, r_{k_{n_k}})\}$, denoting the set of variables x_i that are not allowed to be disclosed to the attacker with precision at least r_i. All the attributes of S_k define one disclosure, i.e., the attacker needs to guess *all* x_i in order to win.

Since a data table in general contains more than one record, x_k corresponds to an entire database column. If we take $d(x_k, x'_k) = \|x_k - x'_k\|_1$, we will compute the worst-case guessing advantage among all table records, assuming that the records are independent from each other. If we take $d(x_k, x'_k) = \|x_k - x'_k\|_\infty$, we will get a stronger result that holds even for correlated records.

3.3.10.1 Guessing a Single Attribute. Let the posterior belief of the adversary be expressed by the probability distribution $\Pr_{post}[\cdot]$. Let the initial distribution of X be $\Pr_{pre}[\cdot]$, and f_X the corresponding probability density function (PDF), i.e. $\Pr_{pre}[X'] = \int_{X'} f_X(x)dx$ for $X' \subseteq X$. Let f_Y be the PDF of the outputs of f_Y. Note that existence of f_X and f_Y is required for the further analysis. We want to compute an upper bound on $\Pr_{post}[X']$ for the set of "sufficiently correct guesses" is defined as $X' = \{x' \mid d(t, x') \leq r\}$ for a given precision radius r, where t is the true input.

We have $\Pr_{post}[X'] := \Pr[x \in X' \mid \mathcal{M}_f(x) = y]$, where y is the output that the attacker observed. Let $\int_A$ denote a Lebesgue integral over a subset A of some metric space. Similarly to [58], we rewrite the

posterior probability as

$$\begin{aligned}
\Pr_{post}[X'] &= \int_{X'} f_X(x' \mid \mathcal{M}_f(x') = y)\, dx' \\
&= \int_{X'} \frac{f_Y(y\,|x')f_X(x')}{f_Y(y)}\, dx' = \frac{\int_{X'} f_Y(y\,|x')f_X(x')\, dx'}{\int_X f_Y(y\,|x)\, f_X(x)\, dx} \\
&= \frac{\int_{X'} f_Y(y\,|x')f_X(x')\, dx'}{\int_{X'} f_Y(y\,|x)\, f_X(x)\, dx + \int_{X\setminus X'} f_Y(y\,|x)\, f_X(x)\, dx} \\
&= \frac{1}{1 + \frac{\int_{X\setminus X'} f_Y(y|x)f_X(x)dx}{\int_{X'} f_Y(y|x')f_X(x')dx'}} \ .
\end{aligned}$$

Let us assume that $R := \sup_{x\in X} d(x, x')$ exists. Differential privacy gives us $\frac{\Pr[\mathcal{M}_f(x')\in Y]}{\Pr[\mathcal{M}_f(x)\in Y]} \leq e^{\epsilon\cdot R}$ for all Y, and hence also $\frac{f_Y(y|x')}{f_Y(y|x)} \leq e^{\epsilon\cdot R}$. We get

$$\Pr_{post}[X'] \leq \frac{1}{1 + e^{-\epsilon R}\cdot \frac{\Pr_{pre}[X\setminus X']}{\Pr_{pre}[X']}} \ .$$

Hence, if we want $\Pr_{post}[X'] \leq \Pr_{pre}[X'] + \epsilon'$, we need to take

$$\epsilon \leq \epsilon_{lb} := \frac{-\ln\left(\frac{\Pr_{pre}[X']}{\Pr_{pre}[X\setminus X']}\cdot\left(\frac{1}{\epsilon'+\Pr_{pre}[X']} - 1\right)\right)}{R} \ .$$

So far, we have shown that $\Pr_{post}[X'] \leq \Pr_{pre}[X'] + \epsilon'$. To satisfy Def. 3.63, we also need $\Pr_{post}[X'] \geq \Pr_{pre}[X'] - \epsilon'$, or $\Pr_{post}[X \setminus X']) \leq \Pr_{pre}[X \setminus X'] + \epsilon'$, to show that the distribution has not changed much in overall compared to prior. The derivation of ϵ for the lower bound is analogous to the upper bound. Substituting $\Pr_{pre}[X']$ with $\Pr_{pre}[X \setminus X']$, we get a bound

$$\epsilon \leq \epsilon_{ub} := \frac{-\ln\left(\frac{\Pr_{pre}[X\setminus X']}{\Pr_{pre}[X']}\cdot\left(\frac{1}{\epsilon'+\Pr_{pre}[X\setminus X']} - 1\right)\right)}{R} \ .$$

To satisfy $\left|\Pr_{post}[X'] - \Pr_{pre}[X']\right| \leq \epsilon'$, we eventually need to take

$$\epsilon = \min(\epsilon_{lb}, \epsilon_{ub}) \ .$$

Note that, if $p \leq 1 - p$, then $\epsilon_{lb} \leq \epsilon_{ub}$, and if $1 - p \leq p$, then $\epsilon_{ub} \leq \epsilon_{lb}$. Hence, we do not need to compute both bounds.

The problem of this approach is that R can be very large in practice, or even not exist. Note that we are essentially trying to prove that the elements of X' are "sufficiently indistinguishable" from the elements of $X \setminus X'$. In practice, it may be sufficient to take just a subset of $\hat{X}' \subseteq X \setminus X'$ and show that it is difficult to distinguish X' and $\hat{X}'$. We have

$$\Pr_{post}[X'] = \frac{1}{1 + \frac{\int_{X\setminus X'} f_Y(y|x)f_X(x)dx}{\int_{X'} f_Y(y|x')f_X(x')dx'}} \leq \frac{1}{1 + \frac{\int_{\hat{X}'} f_Y(y|x)f_X(x)dx}{\int_{X'} f_Y(y|x')f_X(x')dx'}}$$

for any $\hat{X}' \subseteq X \setminus X'$. Let $\hat{X}'_a := \{x \mid d(x, x') \leq a\} \setminus X'$ for some $a \in \mathbb{R}$. We get

$$\epsilon \leq \frac{-\ln\left(\frac{\Pr_{pre}[X']}{\Pr_{pre}[\hat{X}'_a]}\cdot\left(\frac{1}{\epsilon'+\Pr_{pre}[X']} - 1\right)\right)}{a} \ . \tag{36}$$

If $R := \sup_{x \in X, x' \in X'} d(x, x')$ does not exist, then we may instead take a such that $\Pr[x \mid \forall x' \in X' : d(x, x') \leq a] \approx 1$, which is useful e.g. when the input comes from normal distribution. It is possible that there are better candidates for a. We do not have a generic approach for finding an optimal a. In practice, if there is R such that $\Pr[x \mid \forall x' \in X' : d(x, x') \leq R] \approx 1$, since computing ϵ from a is a cheap operation, we may sample several different values of a from the interval $(0, R]$, and take one for which ϵ is the largest, i.e. noise is the smallest.

It remains to show how the values $\Pr_{pre}[X']$ and $\Pr_{pre}[\hat{X}'_a]$ are actually computed. Let g be a function such that $g(z) := \Pr_{pre}[x \mid d(x, t) \leq z]$. The input t is implicitly included into description of g. We have $\Pr_{pre}[X'] = g(r)$ and $\Pr_{pre}[\hat{X}'_a] = g(a) - g(r)$. Let us investigate some special cases of $g(z)$. We also see whether we are able to make any reasonable analysis if prior probabilities are not known in advance, and we look more precisely at discrete datasets.

Uniform distribution. Given a radius r and an upper bound R on $d(x, x')$, it is easy to compute $\Pr_{pre}[X'] = \frac{2r}{R}$. For $a < R$, we can as well take $g(a) = \frac{2a}{R}$. This is true of we can perfectly fit $\hat{X}'_a$ into X. The problem is that the input t may be located not in the "center" of the space X, but somewhere in the corner, as shown in Figure 42. It may happen that we cannot come up with a set $\hat{X}'_a$ with probability weight $\frac{2(a-r)}{R}$ that would satisfy $d(x, x') \leq a$ for all $x \in \hat{X}'_a$ and $x' \in X'$, and need to include more distant points into $\hat{X}'_a$. We may have distance up to $d(x, x') \leq 2a$ for $x' \in X'$, $x \in \hat{X}'_a$, which means that 2 will go to the exponent of e. To avoid the change in exponent, we can just take $a' := 2a$, getting $\Pr_{pre}[X'] = \frac{2r}{R}$ and $g(a') = \frac{a'}{R}$.

Distributions with well-defined cumulative distribution function (CDF). We can estimate noise for normally distributed inputs, and in general for data with well-defined CDF $F(x)$. Namely, since we define $g(z) = \{x | d(x, t) \leq z\}$, then by definition of CDF we get $X' = F(x + z) - F(x - z)$.

If we are not given an upper bound R on X, then we can still derive some R based on the distribution. If the distribution has bell shape, it does not make sense to consider elements that are too far from the center. For example, for normal distribution $N(\mu, \sigma^2)$ we can take $R = \mu + 3 \cdot \sqrt{2}\sigma$, which covers $\mathsf{erf}3 \approx 0.9999779$ of the input space.

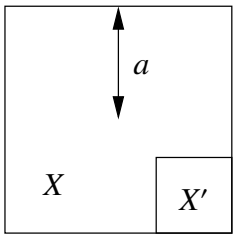


Figure 42: Bad Location of X'

The worst prior probability. Even if the input distribution is unknown, we can still define $\Pr_{pre}[X']$ in such a way that the maximum noise will be needed, so that our analysis would be valid for any possible prior. This value is non-trivial, e.g. although $p = 1.0$ increases posterior probability the most, the advantage would be 0. We want to find p that maximizes

$$\epsilon =: h(p) = \frac{-\ln\left(\frac{p}{1-p} \cdot (\frac{1}{\epsilon'+p} - 1)\right)}{R} \ .$$

We can do it by using common calculus. We have

$$\frac{dh}{dp}(p) = \left(\frac{1}{(p + \epsilon') - (p + \epsilon')^2} - \frac{1}{(1 - p) - (1 - p)^2}\right) \cdot \frac{1}{R} \ ,$$

and we have $\frac{dh}{dp}(p) = 0$ for $p = \frac{1-\epsilon'}{2}$.

Is it reasonable to try out $a < R$ if the input distribution is unknown? In general, if we have no additional knowledge, we only know that $g(R) = 1$. Alternatively, if we do know that $g(a) = q$ for some q, then we can take $q - p$ instead of $1 - p$. For example, even if we do not know details of the input distribution, if it is known that most of the data stays within some $R' < R$, and $\Pr_{pre}[x \mid d(t, x) \leq R'] = q$, it can be useful to give take $a := R'$.

The question is how to choose the worst-case p. Substituting $1-p$ with $g(a)-p$, we get

$$\frac{dh}{dp}(p) = \left(\frac{1}{(p+\epsilon') - (p+\epsilon')^2} - \frac{1}{(q-p)-(q-p)^2}\right) \cdot \frac{1}{R} \ ,$$

and we have $\frac{dh}{dp}(p) = 0$ for $p = \frac{\epsilon'(1-\epsilon')+q(1-q)}{2(\epsilon'+q-1)}$. This is in general different from the $q = 1$ case. However, note that in practice it is fine to consider only the case $q = 1$. If we have found p that minimizes ϵ in the case $a = R$, this ϵ is in any case sufficient. The aim of taking $a < R$ is only to check whether it is possible to come up with a better ϵ. Hence, we can always take $p = \frac{1-\epsilon'}{2}$.

A discrete set of prior probabilities. Suppose that the possible value of x may come from the set of values $\{x_1, \ldots, x_n\}$ with probabilities $\{p_1, \ldots, p_n\}$ respectively ($\sum_{i=1}^n p_i = 1$). Here x is not necessarily a single attribute, and the set $\{p_1, \ldots, p_n\}$ may be computed from all possible combinations of several attributes, which in this case are allowed to be correlated.

If we have access to the actual data, then we may just look at the actual t and take $p = p_k$ such that $t = x_k$, which guarantees that the analysis will be linear w.r.t, the number of table rows. Without having access to the actual data (which may happen if global sensitivity is used to enforce DP), we would need to make a theoretical estimate. We need to take the worst p_k to ensure that each row is protected. If $n \leq \infty$, we can do it for each p_k one by one, but it is not computationally reasonable. Instead, we may start directly from p_k that leaks the most. In the previous paragraph we have shown that the worst case is $p' = \frac{1}{1+e^{-\epsilon/2}}$. This value itself may be missing from $\{p_1, \ldots, p_n\}$. However, since the function $h(p)$ has exactly one local maximum at p' and is monotone at $(-\infty, p']$ and $[p', \infty)$, the values p_l and p_r that are closest to p' from left and right respectively are the worst cases. It suffices to compute ϵ only for these two values, and take the smallest one.

3.3.10.2 Guessing *Each* Sensitive Attribute. Let us compute the ϵ for a single set $S_k \in \mathcal{S}$, where $S_k = \{(t_1, r_1), \ldots, (t_n, r_n)\}$. Assume that there is an upper bound R_i on each dimension. The univariate case can be generalized to multivariate, treating $X = (X_1, \ldots, X_n)$ as a single vector variable, taking $\vec{t} = (t_1, \ldots, t_n)$, $\vec{r} = (r_1, \ldots, r_n)$, $\vec{R} = (R_1, \ldots, R_n)$. For this, we need to clarify how the distance in X should be defined. If it is an ℓ_p norm of underlying dimensions, we will have $d(t, x) = \|t_1 - x_1, \ldots, t_n - x_n\|_p$. According to our intuition, the attacker wins if he guesses correctly a point from the rectangle $[t_1 - r_1, t_1 + r_1] \times \cdots \times [t_n - r_n, t_n + r_n]$. Hence, in general we take $p = \infty$. In some cases, we may as well be interested in other value of p, e.g. if t_1 and t_2 are coordinates of geographic location, then $p = 2$ corresponds more to our intuition.

In Sec. 3.3.10.1, the numbers r and a are single numbers, and not vectors. In general, we should take $R = \|\vec{R}\|_p$, $r = \|\vec{r}\|_p$ and $a = \|\vec{a}\|_p$. However, it is not clear how to come up with $\vec{a}$ in the first place. First of all, let us *scale* all dimensions to get $r := r_i = r_j$ for all i, j, which allows us to define a single r. E.g. take $r = 1$ and scale all R_i accordingly. We can now optimize a based on this r. Still, as the scaled R_i can be different, the same value a might not fit into every dimension. Optimizing a multivariate function by trying out all $a_i \in \{r + \frac{k}{N} \cdot (R_i - r) \mid k \in \{1, \ldots, N\}\}$ is too expensive. Hence, we first sample $r < a \leq R$, which we will use for distance. We will need particular a_i to compute the probability weights $g(a_i)$, and we take $a_i = \min(a, R_i)$.

In Sec. 3.3.10.1, we have shown how to compute $g(z_i)$ for one-dimensional z_i. Assuming that the variables are independent, we can compute $g(z) = \prod_i g(z_i)$ for an AND-event. If the variables are not independent, then there will be some special way of defining $g(z)$, which depends on the distribution. The relations between variables can make only certain combinations of $(a_1, \ldots, a_n)$ possible. However, since a_i are only used in $g(a_i)$ anyway, it suffices to work with a only. Thus, a subspace of correlated variables would be no different from the univariate case, and the probability distribution over that subspace should be given in advance.

The discussion above can be summarized into Theorem 3.88.

Theorem 3.88. *Let $X = (X_1, \ldots, X_n)$, where X_i are pairwise independent, and $R_i = \max_{x_i, x_i' \in X_i} d(x_i, x_i')$. Let $(t_1, \ldots, t_n)$ be the actual data, and let $X' = \{\vec{x} \in X \mid d(t_i, x_i) \leq r_i\}$. Let $g_i(z) := \Pr_{pre}[x_i \mid d(x_i, t_i) \leq z]$. Let $\mathcal{M}_f$ be an ϵ-DP mechanism w.r.t. norm $\|x_1, \ldots, x_n\|_\infty$. To bound the guessing advantage of X' by ϵ', we need to take*

$$\epsilon \leq \frac{-\ln\left(\frac{\prod_{i=1}^n g_i(r_i)}{\prod_{i=1}^n g_i(a_i) - \prod_{i=1}^n g_i(r_i)} \cdot \left(\frac{1}{\epsilon' + \prod_{i=1}^n g_i(r_i)} - 1\right)\right)}{a} .$$

where a_i is a freely chosen value satisfying $r_i < a_i \leq R_i$, and $a = \|a_1, \ldots, a_n\|_\infty$.

3.3.10.3 Guessing *Either* Sensitive Attribute. There are two ways to define guessing advantage for entire $\mathcal{S} = \{S_k \mid k \in [|\mathcal{S}|]\}$:

1. Each $S_k \in \mathcal{S}$ is protected independently. E.g. if we have two uniformly distributed binary variables, then advantage $\epsilon' = 10\%$ means that the probability of guessing increases from 50% to 60% for *any* of these variables. For each $S_k \in \mathcal{S}$, we need to satisfy Def. 3.63 with $X_k = \{x \mid \bigwedge_j x_{i_j} \leq r_{i_j}\}$. The suitable ϵ_k can be found as described in Section 3.3.10.1. Each ϵ_k is converted to noise magnitude λ_k, which tells how much Cauchy noise should be scaled. To protect *each* S_k, we take noise magnitude $\lambda = \max_k(\lambda_k)$.
2. Alternatively, we can estimate the probability that the attacker guesses *at least one* S_k. This poses a different question for an OR condition, and the prior probability gets completely different meaning. E.g. if we have two uniformly distributed binary variables, we will have 75% of correct and 25% of incorrect guesses, and $\epsilon' = 10\%$ means that the probability of guessing *at least one* of these two increases from 75% to 85%. This approach assumes $X_j = \{x \mid \bigvee_k \bigwedge_j x_{i_j} \leq r_{i_j}\}$ in Def. 3.63. In this section, we show how to deal with this variant.

It is easier to estimate approximation of an AND of conditions, since the set X' is bounded by r_i at the coordinate X_i', and $d(x, x') \leq r = \|r_1, \ldots, r_n\|_p$ for all $x, x' \in X$. However, if the attacker wants to guess an OR of conditions, then he may guess x_i correctly even if some other variable x_j is as far from the actual value t_j as possible, and we may only claim that $d(x, x') \leq R = \|R_1, \ldots, R_n\|_\infty$. A comparison of AND and OR sets is depicted in Figure 43a.

If we take $a_i = R_i$ for all i, then the only difference of an OR-event from an AND-event is the construction of $\Pr_{pre}[X']$. We have $\Pr_{pre}[X'] = \prod_{i=1}^n g(a_i) - \prod_{i=1}^n g(a_i - r_i)$.

Let us see what happens if we want to use $a_i < R_i$. Now the problem is that we may have $d(\vec{x}, \vec{x'}) > a$ for $\vec{x}, \vec{x'} \in X'$, as the distances inside X' are not bounded by r. We need to approach it differently.

Let $g_{\vec{t'}}(z) = \Pr[\{x \mid d(x_1, t_1') \leq z \vee \cdots \vee d(x_n, t_n') \leq z\}]$ for $\vec{t'} \in X$. Let $\vec{t} = (t_1, \ldots, t_n)$ be the actual datapoint.

We split both X' and $\hat{X}'$ into the same number of blocks. The idea is that each block X_k' of X' would have a sufficiently close unique neighbor block $\hat{X}'_k$ in $\hat{X}'$. Let $\vec{a} = (a_1, \ldots, a_n)$, and $a = 2\|a_1, \ldots, a_n\|_\infty$. First, we define

$$X_0' := \{\vec{x} \mid \bigwedge_{i=1}^n x_i \in \mathcal{B}(t_i, a_i) \wedge \left(\bigvee_{i=1}^n x_i \in \mathcal{B}(t_i, r_i)\right)\}$$

and

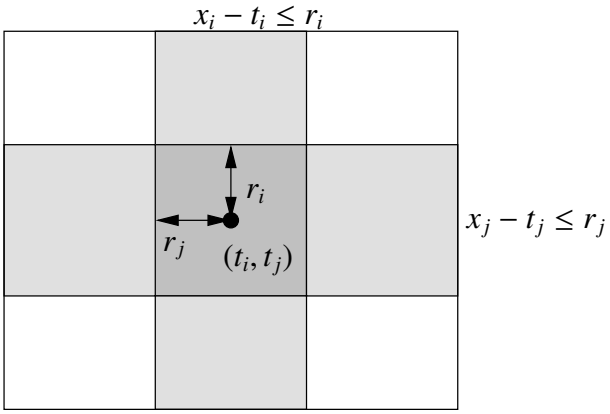
$$\hat{X}_0' := \{\vec{x} \mid \bigwedge_{i=1}^n x_i \in \mathcal{B}(t_i, a_i)\} \setminus X_0' .$$

We also define

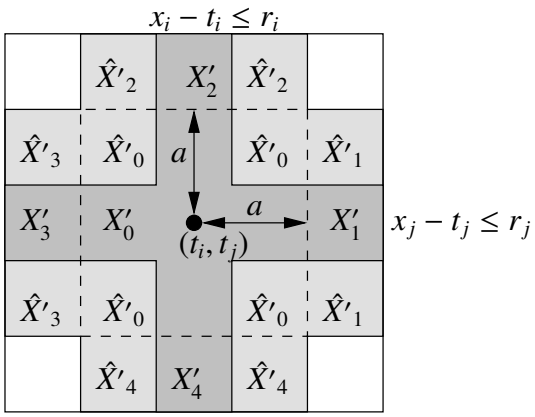
$$X_k^\ell = \{\vec{x} \mid x_k \in \mathcal{B}(t_i, r_i) \wedge \bigwedge_{i=1}^n x_i \in \mathcal{B}(t_i + 2\ell_i \cdot a_i, a_i)\}$$

and

$$\hat{X}_k^\ell = \{\vec{x} \mid \bigwedge_{i=1}^n x_i \in \mathcal{B}(t_i + 2\ell_i \cdot a_i, a_i)\} \setminus \hat{X}_0'$$



(a) The Area of Correctly Guessing AND and OR of Two Approximations



(b) Splitting X' and $\hat{X}'$ Into Easily Comparable Blocks

for $\ell \in \mathcal{L} \subseteq \mathbb{Z}^n$ and $k \in [n]$, where $\mathcal{L}$ depends on the size of the total space X. All blocks are pairwise disjoint, and $X'_0 \cup \bigcup_{\ell,k} X'^{\ell}_k = X'$. An example of splitting into blocks is given in Figure 43b.

The constructed pairs enjoy the following important properties:

1. For all $\vec{x'} \in X'_0$, $\vec{x} \in \hat{X}'_0$ (and also $\vec{x'} \in X^{\ell}_k$, $\vec{x} \in \hat{X}^{\ell}_k$), we have $d(\vec{x}, \vec{x'}) \le a$;
2. We have $\frac{\Pr_{post}[X'_0]}{\Pr_{post}[\hat{X}'_0]} = \frac{g_{\vec{t}}(a_1,\ldots,a_n) - g_{\vec{t}}((a_1-1,\ldots,a_n-1))}{g_{\vec{t}}((a_1-1,\ldots,a_n-1))} =: \alpha_0$;
3. For all $k \in [n]$, $\ell \in \mathcal{L}$, we have $\frac{\Pr_{post}[X^{\ell}_k]}{\Pr_{post}[\hat{X}^{\ell}_k]} = \frac{g_{\vec{t'}}(a_1,\ldots,r_k,\ldots,a_n)}{g_{\vec{t'}}(a_1,\ldots,a_k,\ldots,a_n) - g_{\vec{t'}}(a_1,\ldots,r_k,\ldots,a_n)}$, where $\vec{t'} = (t_1 + 2\ell_1 \cdot a_1, \ldots, t_k, \ldots, t_n + 2\ell_n \cdot a_n)$. Assuming that the variables are independent, this equals $\frac{g_{t_k}(r_k)}{g_{t_k}(a_k) - g_{t_k}(r_k)} =: \alpha_k$.

We will now analyze $\frac{\Pr_{post}[X'_0]}{\Pr_{post}[\hat{X}'_0]}$. By property (1), $d(\vec{x}, \vec{x'}) \le a$, so ϵ-DP gives $f_Y(y \,|\vec{x'}) \le e^{a\epsilon} \cdot f_Y(y \,|\vec{x})$. We have

$$\begin{aligned}\frac{\Pr_{post}[X'_0]}{\Pr_{post}[\hat{X}'_0]} &= \frac{\int_{X'_0} f_Y(y \,|\vec{x'}) \cdot f_X(\vec{x'})\, d\vec{x'}}{\int_{\hat{X}'_0} f_Y(y \,|\vec{x}) \cdot f_X(\vec{x})\, d\vec{x}} \\ &\le e^{a\epsilon} \cdot \frac{\int_{X'_0} f_X(\vec{x'})\, d\vec{x'}}{\int_{\hat{X}'_0} f_X(\vec{x})\, d\vec{x}} = e^{a\epsilon} \cdot \frac{\Pr_{pre}[X'_0]}{\Pr_{pre}[\hat{X}'_0]} = e^{a\epsilon} \cdot \alpha_0 \ .\end{aligned}$$

Similarly, for $k \in [n]$, $\ell \in \mathcal{L}$, we have

$$\frac{\Pr_{post}[X^{\ell}_k]}{\Pr_{post}[\hat{X}^{\ell}_k]} \le e^{a\epsilon} \cdot \frac{\Pr_{pre}[X^{\ell}_k]}{\Pr_{pre}[\hat{X}^{\ell}_k]} = e^{a\epsilon} \cdot \alpha_k \ .$$

Let $\alpha = \max_{k \in \{0,1,\ldots,n\}} \alpha_k$. We get

$$\begin{aligned}\frac{\Pr_{post}[X']}{\Pr_{post}[\hat{X}']} &= \frac{\Pr_{post}[X'_0] + \sum_{k\in[n],\ell\in\mathcal{L}} \Pr_{post}[X^{\ell}_k]}{\Pr_{post}[\hat{X}'_0] + \sum_{k\in[n],\ell\in\mathcal{L}} \Pr_{post}[\hat{X}^{\ell}_k]} \\ &\le \frac{e^{a\epsilon} \cdot \alpha \cdot \Pr_{post}[\hat{X}'_0] + \sum_{k\in[n],\ell\in\mathcal{L}} e^{a\epsilon} \cdot \alpha \cdot \Pr_{post}[\hat{X}^{\ell}_k]}{\Pr_{post}[\hat{X}'_0] + \sum_{k\in[n],\ell\in\mathcal{L}} \Pr_{post}[\hat{X}^{\ell}_k]} \\ &= e^{a\epsilon} \cdot \alpha \ .\end{aligned}$$

The final result depends on $l = \mathsf{argmax}_{k\in[n]} \alpha_k$.

1. Let $l = 0$. By property (2), we have $\frac{\Pr_{pre}[X'_0]}{\Pr_{pre}[\hat{X}'_0]} = \frac{g_i(a_1,\ldots,a_n) - g_i((a_1-1,\ldots,a_n-1)))}{g_i((a_1-1,\ldots,a_n-1))}$. Applying Equation 36, we get some ϵ_0.
2. Let $l = k \neq 0$. By property (3), we have $\frac{\Pr_{pre}[X'^{\ell}_k]}{\Pr_{pre}[\hat{X}'^{\ell}_k]} = \frac{g_{t_k}(r_k)}{g_{t_k}(a_k) - g_{t_k}(r_k)}$ for any $\ell \in \mathcal{L}$. Applying Equation 36, we get some ϵ_k.

We take $\epsilon = \min_{k\in\{0,1,\ldots,n\}}(\epsilon_k)$. This shows that, in addition to protecting each dimension separately, we also need to take care of the $a_1 \times \cdots \times a_n$ block surrounding the actual input $\vec{t}$. We emphasize that it holds only for independent variables, as otherwise we cannot treat different ϵ_k values independently.

The discussion above can be summarized into Theorem 3.89.

Theorem 3.89. *Let $X = (X_1, \ldots, X_n)$, where X_i are pairwise independent, and $R_i = \max_{x_i, x'_i \in X_i} d(x_i, x'_i)$. Let $(t_1, \ldots, t_n)$ be the actual data, and let $X' = \{\vec{x} \in X \mid \bigvee_{i=1}^{n} d(t_i, x_i) \leq r_i\}$. Let $g_i(z) := \max_{t_i \in X'_i} \Pr_{pre}[x_i \mid d(x_i, t_i) \leq z]$. Let $\mathcal{M}_f$ be an ϵ-DP mechanism w.r.t. norm $\|x_1, \ldots, x_n\|_\infty$. To bound the guessing advantage of X' by ϵ', we need to take $\epsilon = \min_{k\in\{0,1,\ldots,n\}}(\epsilon_k)$, where*

$$\epsilon_0 = \frac{-\ln\left(\frac{\prod_{i=1}^{n} g(a_i) - \prod_{i=1}^{n} g(a_i - 1)}{\prod_{i=1}^{n} g(a_i - 1)} \cdot \left(\frac{1}{\epsilon' + \prod_{i=1}^{n} g(a_i) - \prod_{i=1}^{n} g(a_i - 1)} - 1\right)\right)}{a};$$

$$\epsilon_k = \frac{-\ln\left(\frac{g_{t_k}(r_k)}{g_{t_k}(a_k) - g_{t_k}(r_k)} \cdot \left(\frac{1}{\epsilon' + g_{t_k}(r_k)} - 1\right)\right)}{a};$$

where a_i is a freely chosen value satisfying $r_i \leq a_i \leq R_i$, and $a = \|a_1, \ldots, a_n\|_\infty$.

3.3.10.4 Queries with Multiple Outputs. The results of Sec 3.3.10.2 and Sec. 3.3.10.3 do not depend on the definition of the output $f(X)$. They only tell which ϵ is needed to satisfy the desired guessing advantage, and our task is to come up with a DP mechanism achieving this ϵ. The noise should be distributed among multiple outputs in such a way that their composition would be ϵ-DP. For this, we can use the standard notions of parallel and sequential composition.

Theorem 3.90 (Sequential composition [60])**.** *Let $f(x) = (f_1(x) \in \mathbb{R}, \ldots, f_m(x) \in \mathbb{R})$. Let DP mechanism be such that, for all $i \in [m]$, we have $\Pr[\mathcal{M}_{f_i}(x) \in Y] \leq e^{\epsilon_i} \cdot \Pr[\mathcal{M}_{f_i}(x') \in Y]$ for all subsets $Y \subseteq \mathbb{R}$. Then,*

$$\Pr[\mathcal{M}_f(x) \in Y] \leq e^{\sum_{i=1}^{m} \epsilon_i} \cdot \Pr[\mathcal{M}_f(x') \in Y]$$

for all subsets $Y \subseteq \mathbb{R}^m$.

Parallel composition of DP says that we can take $\max(\epsilon_i)$ instead of sum if the variables used by different queries are independent. Roughly speaking, the independence of inputs ensures that the condition $d(t, t') = 1$ affects at most one of the outputs. We can generalize this result using the notion of ℓ_p-norms.

Theorem 3.91. *Let $f(x_1, \ldots, x_n) = (f_1(x_1) \in \mathbb{R}, \ldots, f_m(x_m) \in \mathbb{R})$. Let the distance $d(x, x')$ for $x, x' \in \mathbb{R}^m$ be defined as an ℓ_p-norm. Let a privacy mechanism $\mathcal{M}_{f_i}$ be ϵ_i-differentially-private. Then, the mechanism $\mathcal{M}_f = (\mathcal{M}_{f_1}, \ldots, \mathcal{M}_{f_m})$ is $\ell_q(\epsilon_1, \ldots, \epsilon_m)$-differentially-private, where ℓ_q is the dual norm of ℓ_p.*

Proof. Let $d(x, x') \leq 1$, and let $Y = (Y_1, \ldots, Y_n) \subseteq \mathbb{R}^m$ be arbitrary. We have $x = (x_1, \ldots, x_n)$ and $x' = (x'_1, \ldots, x'_n)$. For an ℓ_p-norm, denoting $d_i := d(x_i, x'_i)$ for all $i \in [m]$, we have $d(x, x') = \|d_1, \ldots, d_m\|_p$. For each i, we have $\Pr[\mathcal{M}_{f_i}(x_i) \in Y_i] \leq e^{d_i\epsilon_i} \cdot \Pr[\mathcal{M}_{f_i}(x') \in Y_i]$. By Theorem 3.90, $\Pr[\mathcal{M}_f(x) \in Y] \leq e^{\sum_{i=1}^{m} d_i\epsilon_i} \cdot \Pr[\mathcal{M}_f(x') \in Y]$, so the mechanism $\mathcal{M}_f$ is $\sum_{i=1}^{m} d_i\epsilon_i$-differentially-private. By definition of the dual norm, $\|\epsilon_1, \ldots, \epsilon_m\|_q = \sup\{\sum_{i=1}^{m} \epsilon_i z_i \mid \|z\|_p \leq 1\}$. Since $(d_1, \ldots, d_m)$ is one candidate for z in this expression, we have $\sum_{i=1}^{m} d_i\epsilon_i \leq \|\epsilon_1, \ldots, \epsilon_m\|_q$. □ □

The most intuitive instantiations for Theorem 3.91 are the cases of ℓ_1 and ℓ_∞ norms, where $d(x, x')$ is an integer value, such as the number of rows. These will give us parallel and sequential compositions. In the case of ℓ_1-norm, $d(x, x') = 1$ means that exactly one of the inputs x_i will change by 1, so it is sufficient to take $\max_i \epsilon_i$ to protect from one change. In the case of ℓ_∞-norm, $d(x, x') = 1$ means that each input x_i may change by 1, so we need to protect all the outputs at once, getting $\sum_i \epsilon_i$.

We can use Theorem 3.91 to find appropriate partitionings to ϵ_i, which are not unique, and the partitioning may be optimized in such a way that the error would be minimal, i.e. reserve larger ϵ_i for more sensitive f_i. Finding the optimal partitioning is a possible direction for future research. If the queries are similar (e.g. groups of a `GROUP BY` query), then the best option is to assign the same value to all ϵ_i.

3.3.10.5 Examples.

Discrete data. Assume that we have a data table `cat`. Each cat can be male or female, and has one of the five main colors. We want to count all female tabby cats.

```
SELECT COUNT(*) FROM cat WHERE gender='F' AND color='tabby';
```

Define the distance between two databases as 1 iff the gender or the color are different. Such distance is subsumed standard DP metric, as every tabby female is viewed as an included record, and every other combination as excluded ,and sensitivity of a COUNT query w.r.t. this metric is 1, similarly to sensitivity of a COUNT query w.r.t. standard DP metric. Hence, the query could be executed with obtained ϵ in PINQ system.

We need to define some prior or the distributions of genders and colors. Let gender be distributed uniformly, and let colors be distributed as

$$\{red : 0.2, white : 0.1, tabby : 0.25, black : 0.4, tortoise : 0.05\} \ .$$

First, assume that the attacker wins if he guesses both gender *and* color. Combining the two sensitive attributes together, we get the probabilities of AND-events

$$\begin{aligned}\{&(red \wedge M) : 0.1, (white \wedge M) : 0.05, (tabby \wedge M) : 0.125, (black \wedge M) : 0.2,\\ &(red \wedge F) : 0.1, (white \wedge F) : 0.05, (tabby \wedge F) : 0.125, (black \wedge F) : 0.2,\\ &(tortoise \wedge M) : 0.025, (tortoise \wedge F) : 0.025\} \ .\end{aligned}$$

We want to bound guessing advantage by $\epsilon' = 0.1$. As shown in Sec. 3.3.10.1, the worst-case prior would be $p = \frac{1-0.1}{2} = 0.45$. The closest to the worst case are $(black \wedge M)$ and $(black \wedge F)$ with $p = 0.2$. We have $p \leq 1 - p$, so it suffices to take

$$\epsilon \leq \frac{-\ln\left(\frac{0.2}{0.8} \cdot (\frac{1}{0.1+0.2} - 1)\right)}{1} = -\ln\left(\frac{0.2}{0.8} \cdot \frac{0.7}{0.3}\right) \approx 0.539 \ .$$

Let us now assume that the attacker wins if he guesses either gender *or* color. This is equivalent to 'not guessing both in a wrong way'. We can write the set of possible correct answers as

$$\begin{aligned}\{&(red \vee M) : 0.6, (white \wedge M) : 0.55, (tabby \wedge M) : 0.625, (black \wedge M) : 0.7,\\ &(red \wedge F) : 0.6, (white \wedge F) : 0.55, (tabby \wedge F) : 0.625, (black \wedge F) : 0.7,\\ &(tortoise \wedge M) : 0.525, (tortoise \wedge F) : 0.525\} \ .\end{aligned}$$

The closest to the worst prior are $(tortoise \wedge M)$ and $(tortoise \wedge F)$ with $p = 0.525$. We have $p \leq 1 - p$, so it suffices to take

$$\epsilon \leq \frac{-\ln\left(\frac{0.475}{0.525} \cdot (\frac{1}{0.1+0.475} - 1)\right)}{1} = -\ln\left(\frac{0.475}{0.425} \cdot \frac{0.425}{0.575}\right) \approx 0.402 \ .$$

As sensitivity of COUNT query w.r.t. proposed metric is 1, using Laplace mechanism (described e.g. in [36]), it would be sufficient to add Laplace noise with scaling parameter $\lambda = 1/0.539 \approx 1.86$ for the AND-event, and $1/0.402 \approx 2.49$ for the OR-event.

Continuous data with univariate approximation. Assume that we have a data table `employee`. Each of the employees has some fixed salary, and we want to find the total employer expenses.

```
SELECT SUM(salary) FROM employee;
```

Assume that the attacker wins if he guesses someone's salary with precision of $r = 100$ currency units. For prior, let us assume that the salary of each employee is distributed normally according to $\mathcal{N}(\mu = 2000, \sigma^2 = 55556)$ (so $\sigma \approx 235.7$), i.e. $\Pr[x \in (1000, 3000)] \approx \mathsf{erf}3 \approx 0.99998$. Using definition of CDF for normal distribution, $\Pr[|x - t| \le r] = \frac{\mathsf{erf}\frac{t+r-\mu}{\sigma\sqrt{2}} - \mathsf{erf}\frac{t-r-\mu}{\sigma\sqrt{2}}}{2} = \frac{\mathsf{erf}\frac{t-1900}{333} - \mathsf{erf}\frac{t-2100}{333}}{2}$. Since there is no particular t available, we consider the worst case.

If we want to bound guessing advantage by $\epsilon' = 0.1$, the worst-case prior would be $p = \frac{1-0.1}{2} = 0.45$. The prior closest to this value is achieved for $t = \mu = 2000$ for $p \approx 0.42$. Let us now take $a = 200$. We have $q = \mathsf{erf}a/(\sigma \cdot \sqrt{2}) \approx 0.6$. Hence,

$$\epsilon \le \frac{-\ln\left(\frac{0.42}{0.6-0.42} \cdot (\frac{1}{0.1+0.42} - 1)\right)}{200} = \frac{0.767}{200} \approx 0.0038 \ .$$

While $\epsilon = 0.0038$ seems extremely small, note that such ϵ would ensure a very strong definition of differential privacy, where neighbour tables are defined as being $a = 200$ units apart. Alternatively, we could rescale the entire space e.g. by $a = 200$, getting $\epsilon = 0.767$. Such ϵ ensures differential privacy for neighbor databases defined as differing in 1 unit. Since each currency unit contributes 1 unit to the sum, the sensitivity of the query w.r.t. latter distance is 1. Laplace noise with scaling $1/\epsilon \approx 1.3$ is sufficient to ensure the guessing advantage below 0.1, and the noise level is comparable to the cat example. We see that dividing ϵ by 200 is fine since we are just working with different magnitudes. Also, such a SUM query would have larger output than a COUNT query (for a similar number of rows), so the noise would have less impact than in the cat example.

Continuous data with multivariate approximation. Assume that we have a data table `ship`. Each ship has some geographic location and certain maximum speed. At some moment, the ships start sailing with their maximum speed towards the port located at the point $(0, 0)$. We want to know when the first ship arrives at the port. Using the operator `<@>` for geographical distance, which is essentially ℓ_2-norm, the query looks like

```
SELECT MIN((POINT(ship.x,ship.y)<@>(0,0))/ship.speed) FROM ship;
```

Assume that the attacker wins if he guesses some ship's location with precision of $r = 10$ units w.r.t. ℓ_2-norm. Let the prior be defined as the distribution of *distances* from the point $(0, 0)$, which is $\mathcal{N}(\mu = 0, \sigma^2 = 1250)$ (so $\sigma \approx 35.4$), i.e. $\Pr[(x, y) \in (-150, 150)] \approx \mathsf{erf}3 \approx 0.99998$. Note that the variables x and y do not need to be independent, as we work with their joint distribution directly.

Let (t_x, t_y) be the actual location of a ship. Unlike the salary example, we can find only a lower bound

$$\Pr[\|(x, y) - (t_x, t_y)\|_2 \le r] \quad \ge \quad \Pr[(\|((x, y) - (0, 0)\|_2 + \|(t_x, t_y) - (0, 0)\|_2 \le r)]$$

$$= \frac{\mathsf{erf}\frac{\|(x,y)\|_2+r-\|(t_x,t_y)\|_2-\mu}{\sigma\sqrt{2}} - \mathsf{erf}\frac{\|(x,y)\|_2-r+\|(t_x,t_y)\|_2-\mu}{\sigma\sqrt{2}}}{2}$$

$$= \frac{\mathsf{erf}\frac{\|(x,y)\|_2-\|(t_x,t_y)\|_2+10}{50} - \mathsf{erf}\frac{\|(x,y)\|_2+\|(t_x,t_y)\|_2-10}{50}}{2} \ .$$

Bounding guessing advantage by $\epsilon' = 0.1$, the worst-case prior would be $p = \frac{1-0.1}{2} = 0.45$. For the lower bound considered above, the prior closest to this value is achieved for $(t_x, t_y) = (0.0)$ for $p \approx 0.22$.

Note that in this case inequality becomes an equality, so p cannot be larger. Let us now take $a = 20$. We have $q = \mathsf{erf}a/(\sigma \cdot \sqrt{2}) \approx 0.43$. Hence,

$$\epsilon \leq \frac{-\ln\left(\frac{0.22}{0.43-0.22} \cdot (\frac{1}{0.1+0.22} - 1)\right)}{10} = \frac{1.25}{20} \approx 0.063 \ .$$

Similarly to the salary example, such ϵ ensures differential privacy where neighbor tables are defined as being $a = 20$ units apart. Scaling the space by by $a = 20$, we get $\epsilon = 1.25$. Let us see how much one distance unit contributes to the final result. If the ship speed is v, then changing its location by 1 will change the arrival time by $\frac{1}{v}$. This will not affect the minimum if that ship was not among the first ones, but in the worst case the query output will also change by $\frac{1}{v}$ as well. Let v be the minimal speed of a ship, e.g. $v = 1$, so that the query sensitivity would be 1 similarly to the previous examples. Laplace noise with scaling $1/\epsilon \approx 0.8$ is sufficient to ensure the guessing advantage below 0.1.

3.3.10.6 Approximated Guessing Advantage. So far, we have shown how to convert ϵ of a DP mechanism to an upper bound ϵ' on guessing advantage. We would like to extend our results to a (ϵ, δ)-DP mechanism. It turns out that, for $\delta > 0$, we cannot guarantee privacy for an arbitrary observed output $y \in Y$. However we can compute a bound on guessing advantage ϵ' that holds with sufficiently high probability δ', similarly to (ϵ, δ)-DP definition. In overall, we get that the guessing advantage is bounded by ϵ' with probability δ', and it is bounded by 1 with probability $1 - \delta'$. That is, in average the guessing advantage is bounded by $\epsilon' \cdot \delta' + (1 - \delta') = 1 + (\epsilon' - 1) \cdot \delta'$, giving us a single number. While providing an bound for *average* case is weaker than providing it for *any* case, we consider it acceptable since the choice of $y \in Y$ is random and is not controlled by the attacker.

Given an (ϵ', δ') requirement on GA, we need to find an appropriate (ϵ, δ) for DP and generate the noise accordingly. The input can be given in terms of average guessing advantage $\Delta = 1 + (\epsilon' - 1) \cdot \delta'$, which gives freedom in choosing ϵ' and δ'. We show that, as far as we can choose ϵ and δ ourselves, for some noise distributions (like Laplace) we can get $\delta' = 0$, which simplifies the solution and allows us to continue using the old definition of GA.

While intuitively (ϵ, δ)-DP means that pure ϵ-DP is satisfied except some bad $y \in Y$ comes into play, this "bad" event probability is not necessarily δ. There are different ways to define (ϵ, δ)-DP, discussed e.g. in [61]. We give transformations between DP and GA for the two main definitions.

Definition 3.64 (Approximate DP [61])**.** Let $\mathcal{M}_f : X \to Y$ be a probabilistic mechanism. We say $\mathcal{M}_f$ is (ϵ, δ) differentially private if for all sets $S \subseteq Y$ and $x_0, x_1 \in X$ we have

$$\Pr[\mathcal{M}_f(x_0) \in S\,] \leq e^{\epsilon \cdot d(x_0, x_1)} \cdot \Pr[\mathcal{M}_f(x_1) \in S\,] + \delta \ .$$

Definition 3.65 (Probabilistic DP [61])**.** Let $\mathcal{M}_f : X \to Y$ be a probabilistic mechanism. We say $\mathcal{M}_f$ is (ϵ, δ)-*probabilistically* differentially private if for all $x_0, x_1 \in X$ there are sets $S_0^\delta, S_1^\delta \subseteq Y$ with $\Pr[\mathcal{M}_f(x_0) \in S_0^\delta] \leq \delta$ and $\Pr[\mathcal{M}_f(x_1) \in S_1^\delta] \leq \delta$ s.t. for all sets $S \subseteq Y$ the following holds:

- $\Pr[\mathcal{M}_f(x_0) \in S \setminus S_0^\delta] \leq e^{\epsilon \cdot d(x_0, x_1)} \cdot \Pr[\mathcal{M}_f(x_1) \in S \setminus S_0^\delta]$;
- $\Pr[\mathcal{M}_f(x_1) \in S \setminus S_1^\delta] \leq e^{\epsilon \cdot d(x_0, x_1)} \cdot \Pr[\mathcal{M}_f(x_0) \in S \setminus S_1^\delta]$.

It is easier to achieve (ϵ, δ)-GA for Definition 3.65. Let $y \in Y$ be such that $f_Y(y|x) \leq f_Y(y|x') \cdot e^{\epsilon \cdot d(x,x')}$ for all x, x'. By Definition 3.65, this happens for at least $1 - \delta$ of values $y \in Y$, so we now use the same δ in GA and DP definitions. For those y, we compute ϵ in the same way as we did it for ϵ-DP. The desired guessing advantage is achieved with probability $1 - \delta$.

Definition 3.64 does not give a direct conversion to probability of failure. However, a good thing is that many standard mechanisms that satisfy Definition 3.64 (Gaussian [62], truncated Laplace [57]) also satisfy $f_Y(y|x) \leq f_Y(y|x') \cdot e^{\epsilon \cdot d(x,x')}$ with probability $1 - \delta$, treating y as a random variable for fixed x, x'. The question is whether we can get the same set of "good" values y for *all* $x' \in X$. The set of "bad" values y comes from the tails of these distributions, and the set of "good" values of y obtained for particular (x, x') such that $d(f(x), f(x'))$ is the maximum possible distance between any two outputs,

will also be "good" for any closer x and x'. However, for n outputs, we cannot just apply sequential composition of DP δ values, but instead can only take $(1-\delta)^n$, i.e. probability that $(y_1, \ldots, y_n)$ is a vector of "all good" values.

Theorem 3.63 of Section 3.3.8, describes how to use derivative sensitivity to sample Laplace noise to achieve $(\epsilon, \delta) - DP$. However, we cannot claim that $f_Y(y|x) \leq f_Y(y|x') \cdot e^{\epsilon \cdot d(x,x')}$ holds with probability at least $1 - \delta$. Let us try to analyze the Laplace distribution directly, as in [58].

Analysis based on global sensitivity. Let λ be the scaling parameter of Laplace distribution. We have $\frac{f_Y(y|x)}{f_Y(y|x')} \leq e^{\frac{d(f(x),f(x'))}{\lambda}} \leq e^{\frac{K}{\lambda}}$, where $K = \max_{x,x' \in X} |f(x) - f(x')|$ is the maximum possible difference between outputs. On the other hand,for a generic $(\epsilon, 0)$-DP mechanism, we would get $e^{\epsilon \cdot R}$. The appropriate scaling λ of Laplace distribution will thus be $\lambda \geq \frac{K}{\epsilon \cdot R}$, where ϵ would be DP parameter required to satisfy δ'-GA if $(\epsilon, 0)$-DP was used. The quantity K can now be estimated in different ways. A valid upper bound is $GS_f \cdot R$, where $R = \max_{x,x' \in X} d(x, x')$, but it can be quite rough. For example, if $f(x) = x^2$ for $x \in [0, 10]$, we get $GS_f = \max_{x,x' \in [0,10], |x-x'|=1} \left|x^2 - x'^2\right| = 19$, but $190 = 19 \cdot 10$ is almost 2 times larger than the actual bound $K = \max_{x,x' \in [0,10]} \left|x^2 - x'^2\right| = 100$. In Section 3.3.11, we discuss other methods for approximating K.

Analysis based on derivative sensitivity. In some cases, global sensitivity does not give a reasonable noise magnitude. Let us estimate $K = \max_{x,x' \in X} |f(x) - f(x')|$ using derivative sensitivity. By mean value theorem (Theorem 3.61), there exists $t^* \in (0, 1)$ such that $|f(x) - f(x')| \leq \|df_z\| \|x - x'\|$ for $z = (1 - t^*)x + t^* x'$. Let now c_β be a β-smooth upper bound on DS_f. We have

$$\left|f(x) - f(x')\right| \leq c_\beta(z) \|x - x'\| \ ,$$

and by β-smoothness of c_β,

$$\left|f(x) - f(x')\right| \leq c_\beta(x) \cdot e^{\beta \cdot \|x-z\|} \|x - x'\| \ ,$$

which gives $K \leq c_\beta(x) \cdot e^{\beta \cdot R} \cdot R$.

Using Theorem 3.63, we can convert the obtained noise magnitude λ to suitable ϵ and δ for differential privacy. Theorem 3.63 allows some flexibility when choosing these parameters, and the partitioning to ϵ and δ that corresponds to the same scaling λ will not be unique.

3.3.10.7 Attackers with Partial Knowledge. So far, we considered the worst case of DP, where the attacker already knows the values of all other records except the one he is trying to guess. The proposed upper bound on posterior guessing probability ranges from 0 to 1 regardless of the value of the true output (i.e. without noise). Alternatively, we could assume an attacker who knows less. For example, if the attacker sees the output of a count query, even without noise he may be unable to infer whether some particular row is in the table, if he does not know anything about the other rows. This would allow to achieve smaller bounds on guessing advantage.

In this section, since we reserve the notation f for probability density functions, we denote the query function by q.

Let the conditional probability of having the input $x \in X$ after observing the output $z \in Z := q(X)$ (without noise) be characterized by a probability density function $f_X(x|z)$. Assume that we know how to compute $f_X(x|z)$. In general, this is already a non-trivial task that depends on the function and the distribution of X, and is not directly composable. A simple example is a count-query where each row is equally likely present in the table, and there are up to m available rows. In this case, we have $f_X(x|z) = \frac{z}{m}$ for all $x \in X$.

Let f_Z be the probability density function over true outputs z. We may express the probability of guessing the input x from the output y as

$$f_X(x|y) = \int_{z \in Z} f_X(x|z, y) f_Z(z|y) dz \ .$$

Since the likelihood of x depends directly on z, and y is just a noisy version of z, we have $f_X(x|z,y) = f_X(x|z)$. We can compute $f_Z(z|y)$ using Bayesian inference as $f_Z(z|y) = \frac{f_Y(y|z)\cdot f_Z(z)}{f_Y(y)}$, where $f_Y(y|z)$ is defined by the particular noise distribution, and $f_Y(y) = \int_{z\in Z} f_Z(z)\cdot f_Y(y|z)\,dz$. This gives

$$f_X(x|y) = \frac{\int_{z\in Z} f_X(x|z) f_Y(y|z) f_Z(z) dz}{\int_{z\in Z} f_Y(y|z) f_Z(z) dz} \quad . \tag{37}$$

While integration over Z is difficult and we do not know how to do it in general, for discrete Z we can do it by replacing $\int_{z\in Z}$ with $\sum_{z\in Z}$.

Computation of $f_Z(z)$ is another non-trivial task that depends on the computed function. For a count-query over a table which is known to contain up to m records, and the i-th row is present in the table with probability p_i, we have $f_Z(z) = \sum_{\mathcal{I}\subseteq[m]} \prod_{i\in\mathcal{I}} p_i \cdot \prod_{i\notin\mathcal{I}}(1-p_i)\cdot[q(\mathcal{I}) = z]$, where $[q(\mathcal{I}) = z] \in \{0,1\}$ is defined as 1 iff $q(T) = z$ for T such that exactly the records indexed by $\mathcal{I}$ are in the table T. If each row is equally likely present in the table (with probability p), we have $f_Z(z) = \binom{m}{z}\cdot p^z \cdot (1-p)^{m-z}$.

Finally, we need the quantity $f_Y(y|z)$, which depends on the particular privacy mechanism in use. Ideally, we want to compute a bound that would purely be implied by definition of DP, without going into details of distribution $f_Y(y|z)$. Intuitively, we want that this bound would depend on $q(x)$, and the attacker is not able to learn more about x than it would from the true output $q(x)$. We will see whether it is always possible.

Limitations of using general DP definition. Let us show that we do not get an upper bound $f_X(x|q(x))$ if we are given *only* the definition of DP without specifying anything about the used DP mechanism. The main reason is that DP only tells us something about the distribution of *noisy* outputs $\mathbf{Y}$, but nothing about the distribution of *true* outputs $\mathbf{C}^r$, which may be completely different algebraic structures. The next example shows a DP mechanism in which probability of guessing x does not depend on q at all. Let $\mathbf{X}$ be the random variable corresponding to the true input x.

Proposition 3.92. *Let $\mathcal{M}_q : X \to \mathbb{R}$ be defined as $\mathcal{M}_q(x) = a$ for some constant $a \in \mathbb{R}$. The following is true about $\mathcal{M}_q$:*

1. *$\mathcal{M}_q$ is ϵ-differentially private for any ϵ;*
2. *$\Pr[q(x) = z|\mathcal{M}_q(x) = y] = \Pr[q(x) = z]$ for all $y \in Y, z \in q(X)$.*

Proof. For all $x, x\ \in X$, we have

$$\begin{aligned}\Pr[\mathcal{M}_q(x) \in Y|\mathbf{X} = x] &= \Pr[a \in Y] \\ &= \Pr[\mathcal{M}_q(x') \in Y|\mathbf{X} = x'] \\ &\leq \Pr[\mathcal{M}_q(x') \in Y|\mathbf{X} = x'] \cdot e^{\epsilon\|x-x'\|}\end{aligned}$$

for any $\epsilon > 0$, so $\mathcal{M}_q$ is ϵ-differentially private for any ϵ. Similarly, $\Pr[q(x) = z|\mathcal{M}_q(x) = y] = \Pr[q(x) = z|a] = \Pr[q(x) = z]$. □

Indeed, the example of Proposition 3.92 would not be very useful for utility, since $\mathcal{M}_q$ is a constant. However, the pure definition of DP does not say anything about the utility. We show that, even if we add notion of utility, it will still be not enough. In the next example, utility increases as noise magnitude decreases, but the upper bound on probability of guessing x still does not depend on $q(x)$.

Proposition 3.93. *Let $X = \{0,1\}$. Let $\mathcal{M}_q : X \to \mathbb{R}$ be defined as $\mathcal{M}_q(x) = x + \eta$, where $\eta \leftarrow Lap(\frac{1}{\epsilon})$ for some $\epsilon > 0$. The following is true about $\mathcal{M}_q$:*

1. *$\mathcal{M}_q$ is ϵ-differentially private;*
2. *there exists ϵ such that $\Pr[\mathbf{X} = x|\mathbf{Y} = \mathcal{M}_q(x)] > \Pr[\mathbf{X} = x|\mathbf{C}^r = q(x)]$ for all $x \in X$.*

Proof. We can view $\mathcal{M}_q$ as applying Laplace mechanism to an identity function $q'(x) = x$, which can in turn be viewed as a COUNT query applied to a table that contains either 0 or 1 rows. Since we know that adding noise sampled from $Lap(\frac{1}{\epsilon})$ is ϵ-private w.r.t. a COUNT query, we have $\Pr[\mathcal{M}_q(x) \in Y|\mathbf{X} = x] = \Pr[\mathcal{M}_{q'}(x) \in Y|\mathbf{X} = x] \leq \Pr[\mathcal{M}_{q'}(x) \in Y|\mathbf{X} = x'] \cdot e^{\epsilon\|x-x'\|} = \Pr[\mathcal{M}_q(x) \in Y|\mathbf{X} = x'] \cdot e^{\epsilon\|x-x'\|}$, so $\mathcal{M}_q$ is ϵ-differentially private.

The second property follows from the fact that the noisy output does not depend on $q(x)$ at all. More formally, instantiating (37) with identity function, we get

$$\begin{aligned}
\Pr[\mathbf{X} = x|\mathbf{Y} = \mathcal{M}_q(x)] &= \frac{\int_{x' \in X} f_X(x|x') f_Y(y|x') f_X(x') dx'}{\int_{x \in X'} f_Y(y|x') f_X(x') dx} \\
&= \frac{f_X(x|x) f_Y(y|x) f_X(x) + f_X(x|x') f_Y(y|x') f_X(x')}{f_Y(y|x') f_X(x') + f_Y(y|x) f_X(x)} \\
&= \frac{f_Y(y|x) f_X(x)}{f_Y(y|x') f_X(x') + f_Y(y|x) f_X(x)} = \frac{1}{\frac{f_Y(y|x') f_X(x')}{f_Y(y|x) f_X(x)} + 1} \\
&= \frac{1}{e^{-\epsilon(|y-x|-|y-x'|)} \cdot \frac{1-f_X(x)}{f_X(x)} + 1} = \frac{1}{e^{-\epsilon} \cdot \frac{1-f_X(x)}{f_X(x)} + 1} \ .
\end{aligned}$$

We can take ϵ as large as we want to make the result arbitrarily close to 1, so $\Pr[\mathbf{X} = x|\mathbf{C}^r = q(x)]$ would not be a valid upper bound on guessing probability, unless q itself is an identity function. □

In the example of Proposition 3.93, we have not lost the utility, and the noisy answer does contain whole information about x. However, now it may contain too much information about x, and we can no longer claim that it leaks at most as much as $q(x)$. We see that we need to define an additional property, which has to be satisfied by a large class of standard DP mechanisms that we want to apply.

DP mechanisms with special properties. We give a more constrained definition of a DP mechanism that allows to bound posterior guessing probability with $f_X(x|q(x))$. Let $\mathbf{C}^g$ be the guess about $q(\mathbf{X})$ that the attacker makes after observing $\mathcal{M}_q(x)$ (to distinguish it from the actual true output $\mathbf{C}^r$).

Definition 3.66. An ϵ-DP mechanism $\mathcal{M}_q$ is called *output-bounded* if for all $x \in X'$:

$$\lim_{\epsilon \to \infty} f_Z(\mathbf{C}^g = q(x)|\mathbf{X} = x) = 1 \ .$$

Intuitively, Def. 3.66 says that, the less noise the given DP mechanism gives, the more likely attacker thinks that the true output should be $q(x)$. Examples of such mechanisms would be any additive noise with a bell-shaped distribution, such as Cauchy, Laplace, Gaussian distributions. The limit corresponds to narrowing the bell curve.

We are also interested in the limit $\epsilon \to 0$. It turns out that DP definition is enough. Let us first prove a small lemma that we will use later.

Lemma 3.94. *Let $\mathcal{M}_q$ be any ϵ-differentially private mechanism. For all $x, x' \in X$ we have*

$$f_Z(\mathbf{C}^g = z|\mathbf{X} = x) \leq f_Z(\mathbf{C}^g = z|\mathbf{X} = x') \cdot e^{\epsilon\|x-x'\|} \ .$$

Proof. Note that the value x is not observed directly by the attacker. Instead, he sees a noised value y that in turn depends on x. We can write it out as

$$f_Z(\mathbf{C}^g = z|\mathbf{X} = x) = \int_{y \in Y} f_Z(\mathbf{C}^g = z|Y = y, \mathbf{X} = x) f_Y(y|\mathbf{X} = x) dy \ .$$

The attacker only sees $\mathbf{Y}$ and does not use $\mathbf{X}$ directly, so $f_Z(\mathbf{C}^g = z|\mathbf{Y} = y, \mathbf{X} = x) = f_Z(\mathbf{C}^g = z|\mathbf{Y} = y) = f_Z(\mathbf{C}^g = z|\mathbf{Y} = y, \mathbf{X} = x')$ for any input x'. The definition of DP gives us $f_Y(y|\mathbf{X} = x) \leq f_Y(y|\mathbf{X} = x') \cdot e^{\epsilon\|x-x'\|}$. We can write

$$
\begin{aligned}
f_Z(\mathbf{C}^g = z|\mathbf{X} = x) &\leq \int_{y\in Y} f_Z(\mathbf{C}^g = z|\mathbf{Y} = y, \mathbf{X} = x')f_Y(y|\mathbf{X} = x') \cdot e^{\epsilon\|x-x'\|}dy \\
&= \int_{y\in Y} f_Z(\mathbf{C}^g = z|\mathbf{Y} = y)f_Y(y|\mathbf{X} = x') \cdot e^{\epsilon\|x-x'\|}dy \\
&= f_Z(\mathbf{C}^g = z|\mathbf{X} = x') \cdot e^{\epsilon\|x-x'\|} \ .
\end{aligned}
$$

□

It is now straightforward to prove the following proposition.

Proposition 3.95. *Let $\mathcal{M}_q$ be any ϵ-differentially private mechanism. Let $\mathbf{C}^g$ be the guess about $q(x)$ that the attacker makes after observing $\mathcal{M}_q(x)$. We have*

$$
\lim_{\epsilon\to 0} f_Z(\mathbf{C}^g = q(x)|\mathbf{X} = x) = f_Z(\mathbf{C}^g = q(x)) \ .
$$

Proof. By Lemma 3.94, we know that $f_Z(\mathbf{C}^g = z|\mathbf{X} = x) \leq f_Z(\mathbf{C}^g = z|\mathbf{X} = x') \cdot e^{\epsilon\|x-x'\|}$ for all $x, x' \in X$. We have $\lim_{\epsilon\to 0} e^{\epsilon\|x-x'\|} = 1$, so $\lim_{\epsilon\to 0} f_Z(\mathbf{C}^g = q(x)|\mathbf{X} = x) = f_Z(\mathbf{C}^g = q(x)|\mathbf{X} = x')$. Since it holds for all $x, x' \in X$, the variable $\mathbf{C}^g$ does not depend on the choice of x, and we have $\lim_{\epsilon\to 0} f_Z(\mathbf{C}^g = q(x)|\mathbf{X} = x) = f_Z(\mathbf{C}^g = q(x))$. □

We want to show that an output-bounded privacy mechanism gives us the desired limits for posterior guessing advantage.

If we are not given a particular $y \in Y$ and want to know how successful the attacker is in average, we need to compute $f_X(x|c) = \int_Y f_X(x|y) \cdot f_Y(y|c)dy$. This integration is complicated, as we do not even have a closed form for $f_X(x|y)$. We want to simplify this computation instead of integrating over Y.

Choosing $y = c$ gives us a fair comparison with the solution that uses no privacy mechanisms, as it compares how well the secret can be guessed from seemingly similar output. This works well for DP, but with other privacy mechanisms it is possible that $c \notin \mathcal{M}_q(X)$. Another important property of $y = c$ is that it gives the highest value for $f_Y(y|c)$, so it can be viewed as the most probable outcome, and it is also the noised value that we get in average. However, average noise does not necessarily give us the average posterior guessing probability. We now generalize our previous results to three different levels of guessing: $P_Y(x, y, \mathsf{A})$ based on particular noisy output y, $P_C(x, c, \mathsf{A})$ based on particular true output c, and $P_P(x, \mathsf{A})$ for an average distribution of outputs, where A is auxiliary knowledge that the attacker may possess.

Posterior for a particular noisy output instance. Following the idea of estimating the likelihood of x from all possible values of true outputs z, we can write out attacker's posterior guessing probability as

$$
\begin{aligned}
P_Y(x, y, \mathsf{A}) &:= f_X(x|\mathbf{Y} = y, \mathsf{A}) \\
&= \int_C f_X(x|y, \mathbf{C}^g = z, \mathsf{A}) \cdot f_Z(z|y, \mathsf{A})\, dz \ .
\end{aligned}
$$

After the attacker has selected a true output z according to the noisy output y, it makes its guess purely from z and the additional knowledge A, so $f_X(x|\mathbf{Y} = y, \mathbf{C}^g = z, \mathsf{A}) = f_X(x|\mathbf{C}^g = z, \mathsf{A})$. To estimate $f_Z(z|\mathbf{Y} = y, \mathsf{A})$, the attacker takes into account the likelihood of the noise that would turn z into y, as well as the probability of z itself. It can be done using Bayesian inference as $f_Z(z|\mathbf{Y} = y, \mathsf{A}) = \frac{f_Y(y|\mathbf{C}^g=z,\mathsf{A})\cdot f_Z(z \mid \mathsf{A})}{f_Y(y \mid \mathsf{A})}$. Since A only contains knowledge about the data, and not the distribution, we have $f_Y(y|\mathbf{C}^g = z, \mathsf{A}) = f_Y(y|\mathbf{C}^g = z)$, which can be computed from the noise function and the point z. Let its PDF be denoted $f_{YC}(y, z) := f_Y(y|\mathbf{C}^g = z)$. The quantity $f_Z(z\,|\mathsf{A})$ can be computed from prior probabilities, taking into account the additional knowledge. From these two quantities, we can in turn compute $f_Y(y\,|\mathsf{A}) = \int_C f_Y(y|\mathbf{C}^g = z)f_Z(z\,|\mathsf{A})dz$. We get

$$
P_Y(x, y, \mathsf{A}) = \frac{\int_C f_X(x|\mathbf{C}^g = z, \mathsf{A}) \cdot f_{YC}(y, z)f_Z(z \mid \mathsf{A})\, dz}{\int_C f_{YC}(y, z)f_Z(z \mid \mathsf{A})\, dz} \ . \tag{38}
$$

Posterior for a particular true output instance. Fixing particular $y \in Y$ may make the attacker seem too successful or too unlucky, depending on the $y \in Y$ that we have got. Knowing a particular distribution on Y, we may estimate how much the attacker may guess in average for a particular output $c := q(x) \in C$.

$$\begin{aligned} P_C(x, c, \mathsf{A}) &:= f_X(x|\mathbf{C}^r = c, \mathsf{A}) \\ &= \int_Y f_X(x|y, \mathbf{C}^r = c, \mathsf{A}) f_Y(y|\mathbf{C}^r = c, \mathsf{A})\, dy \\ &= \int_Y f_X(x|y, \mathsf{A}) f_Y(y|\mathbf{C}^r = c, \mathsf{A})\, dy \\ &= \int_Y P_Y(x, y, \mathsf{A}) \cdot f_Y(y|\mathbf{C}^r = c, \mathsf{A})\, dy \ . \end{aligned}$$

The quantity $f_Y(y|\mathbf{C}^r = c, \mathsf{A}) = f_Y(y|\mathbf{C}^r = c)$, which is the likelihood of turning c into y, can be computed from the noise function, similarly to $f_Y(y|\mathbf{C}^g = z)$. We get

$$P_C(x, c, \mathsf{A}) = \int_Y P_Y(x, y, \mathsf{A}) \cdot f_{YC}(y, c)\, dy \ . \tag{39}$$

In practice, we may want privacy for *any* possible outcome $y \in Y$. Differential privacy holds for all $y \in Y$ by definition, so we wonder if we get similar guarantees for guessing advantage. Given c, we can be interested in the worst y that helps in guessing input the most. Such worst-case y should not depend on the particular input x, as in general we can always come up with y for which the true x is the most possible, making the attacker's guessing advantage seem too good, so we would need to find average over possible inputs x, or use some other approach. We propose that taking average over $f_Y(y \mid c)$ gives a more reasonable result.

Posterior for a particular prior. Applying our analysis to a particular testing set still gives us results that depend on the data. If we know the *distribution* of data, we can estimate the posterior probabilities not for the particular counts, but in average. We will still assume a fixed total number m of users. Let $f_C(\cdot)$ be the PDF of actual outputs, and let $C := Z$ be the set of possible true outputs. Note that it is possible that $f_C(\cdot) \neq f_Z(\cdot)$, as $f_Z(\cdot)$ describes attacker's knowledge about the input, which may be different. We have

$$\begin{aligned} P_P(x, \mathsf{A}) &:= f_X(x \mid \mathbf{X} = x, \mathsf{A}) \\ &= \int_C f_X(x|\mathbf{C}^r = c, \mathsf{A}) f_C(c|x) dc \\ &= \int_C P_C(x, c, \mathsf{A}) \cdot f_C(c|x)\, dc \\ &= \int_C \left(\int_Y P_Y(x, y, \mathsf{A}) \cdot f_{YC}(y, c)\, dy \right) f_C(c|x)\, dc \\ &= \int_Y P_Y(x, y, \mathsf{A}) \cdot \left(\int_C f_{YC}(y, c) f_C(c|x)\, dc \right) dy \ . \end{aligned}$$

Since $f_C(c|x) = \frac{f_X(x|\mathbf{C}^r = c) \cdot f_C(c)}{f_X(x)}$, we get

$$P_P(x, \mathsf{A}) = \int_Y P_Y(x, y, \mathsf{A}) \cdot \frac{\int_C f_X(x|c) f_{YC}(y, c) f_C(c)\, dc}{f_X(x)} dy \ , \tag{40}$$

which we may as well express through $P_C(x, c, \mathsf{A})$ as

$$P_P(x, \mathsf{A}) = \int_C P_C(x, c, \mathsf{A}) \cdot \frac{f_X(x|c) \cdot f_C(c)}{f_X(x)}\, dc \ . \tag{41}$$

Let us instantiate the latter quantity to the case when there is no privacy mechanism in use. If the attacker sees that the outcome is c, the probability of x being there is $f_X(x \mid c, \mathsf{A})$. We need to iterate through all possible outcomes c on condition that $\mathbf{X} = x$, getting

$$f_X(x \mid \mathsf{A}) = \frac{1}{f_X(x)} \cdot \int_C f_X(x \mid \mathbf{C}^r = c, \mathsf{A}) f_X(x \mid \mathbf{C}^r = c) f_C(c)\, dc \tag{42}$$

If $\mathsf{A} = \emptyset$, then $f_X(x \mid \mathbf{C}^r = c, \mathsf{A}) = f_X(x \mid \mathbf{C}^r = c)$, and this probability is very close to the prior. If we did not have the condition $\mathbf{X} = x$, then it would be exactly the prior. In general, this result shows that posterior will in average not be far from prior, at least as far as the attacker is not given any additional knowledge. This would mean that DP does not make much sense against such weak attacker. However, the initial idea of DP is to protect the data against attackers who already know something. In our experiments, we compute advantage of different attackers with different additional knowledge.

Limits. Intuitively, we want that the posterior guessing probability converged to $f_X(x)$ as $\epsilon \to 0$, and to $f_X(x|\mathbf{C}^g = q(x))$ as $\epsilon \to \infty$. We will prove that the same holds for $P_Y(x, y, \mathsf{A})$, $P_C(x, c, \mathsf{A})$, $P_P(x, \mathsf{A})$.

Theorem 3.96. *Let an output-bounded ϵ-DP mechanism be in use. We have*

1. $\lim_{\epsilon\to 0} P_Y(x, y, \mathsf{A}) = f_X(x \mid \mathsf{A})$,
   $\lim_{\epsilon\to\infty} P_Y(x, y, \mathsf{A}) = f_X(x \mid \mathbf{C}^g = y, \mathsf{A})$.
2. $\lim_{\epsilon\to 0} P_C(x, c, \mathsf{A}) = f_X(x \mid \mathsf{A})$,
   $\lim_{\epsilon\to\infty} P_Y(x, c, \mathsf{A}) = f_X(x \mid \mathbf{C}^g = c, \mathsf{A})$.
3. $\lim_{\epsilon\to 0} P_P(x, \mathsf{A}) = f_X(x \mid \mathsf{A})$,
   $\lim_{\epsilon\to\infty} P_P(x, \mathsf{A}) = \frac{1}{f_X(x)} \cdot \int_C f_X(x \mid \mathbf{C}^r = c, \mathsf{A}) f_X(x \mid \mathbf{C}^r = c) f_C(c)\, dc$.

Proof. First of all, as the privacy mechanism is output-bounded, we have the following limits.

- $\lim_{\epsilon\to\infty} f_Y(y \mid \mathbf{C}^g = c) = 1$, and $\lim_{\epsilon\to 0} f_Y(y \mid \mathbf{C}^g = z) = 0$ for $z \neq c$.
- $\lim_{\epsilon\to 0} f_Y(y \mid \mathbf{C}^g = z) = f_Y(y)$ for all y, z.

The first one is a property of output-bounded mechanism, and it follows directly from Definition 3.66. The second property holds for any differentially private mechanism, as follows from Proposition 3.95.

We prove theorem statements one by one. Take the definition of $P_Y(x, y, \mathsf{A})$ from Equation 38.

$$\begin{aligned}
\lim_{\epsilon\to 0} P_Y(x, y, \mathsf{A}) &= \lim_{\epsilon\to 0} \frac{\int_C f_X(x|\mathbf{C}^g = z, \mathsf{A}) \cdot f_{YC}(y, z) f_Z(z \mid \mathsf{A})\, dz}{\int_C f_{YC}(y, z) f_Z(z \mid \mathsf{A})\, dz} \\
&= \frac{\int_C f_X(x|\mathbf{C}^g = z, \mathsf{A}) \cdot f_Y(y) \cdot f_Z(z \mid \mathsf{A})\, dz}{\int_C f_Y(y) \cdot f_Z(z \mid \mathsf{A})\, dz} \\
&= f_X(x \mid \mathsf{A}) \ .
\end{aligned}$$

$$\begin{aligned}
\lim_{\epsilon\to\infty} P_Y(x, y, \mathsf{A}) &= \lim_{\epsilon\to\infty} \frac{\int_C f_X(x|\mathbf{C}^g = z, \mathsf{A}) \cdot f_{YC}(y, z) f_Z(z \mid \mathsf{A})\, dz}{\int_C f_{YC}(y, z) f_Z(z \mid \mathsf{A})\, dz} \\
&= \frac{f_X(x|\mathbf{C}^g = y, \mathsf{A}) \cdot f_{YC}(y, y) f_Z(y \mid \mathsf{A})}{f_{YC}(y, y) f_Z(y \mid \mathsf{A})} \\
&= f_X(x|\mathbf{C}^g = y, \mathsf{A}) \ .
\end{aligned}$$

Take the definition of $P_C(x, c, \mathsf{A})$ from Equation 39. We make use of the proven limits for $P_Y(x, y, \mathsf{A})$.

$$\begin{aligned}\lim_{\epsilon\to 0} P_C(x,c,\mathsf{A}) &= \lim_{\epsilon\to 0}\int_Y P_Y(x,y,\mathsf{A})\cdot f_{YC}(y,c)\,dy\\ &= f_X(x\mid\mathsf{A})\cdot\int_Y f_{YC}(y,c)\,dy\\ &= f_X(x\mid\mathsf{A})\ .\end{aligned}$$

$$\begin{aligned}\lim_{\epsilon\to\infty} P_C(x,c,\mathsf{A}) &= \lim_{\epsilon\to\infty}\int_Y P_Y(x,y,\mathsf{A})\cdot f_{YC}(y,c)\,dy\\ &= \int_Y f_X(x|\mathbf{C}^g=y,\mathsf{A})f_{YC}(y,c)\,dy\\ &= f_X(x|\mathbf{C}^g=c,\mathsf{A})\ .\end{aligned}$$

Take the definition of $P_P(x,\mathsf{A})$ from Equation 41. We make use of the proven limits for $P_Y(x,c,\mathsf{A})$.

$$\begin{aligned}\lim_{\epsilon\to 0} P_P(x,\mathsf{A}) &= \int_C P_C(x,c,\mathsf{A})\cdot\frac{f_X(x|c)\cdot f_C(c)}{f_X(x)}\,dc\\ &= f_X(x\mid\mathsf{A})\int_C\frac{f_X(x|c)\cdot f_C(c)}{f_X(x)}\,dc\\ &= f_X(x\mid\mathsf{A})\ .\end{aligned}$$

$$\begin{aligned}\lim_{\epsilon\to\infty} P_P(x,\mathsf{A}) &= \int_C P_C(x,c,\mathsf{A})\cdot\frac{f_X(x|c)\cdot f_C(c)}{f_X(x)}\,dc\\ &= \int_C f_X(x|\mathbf{C}^g=c,\mathsf{A})\frac{f_X(x|c)\cdot f_C(c)}{f_X(x)}\,dc\\ &= \frac{1}{f_X(x)}\cdot\int_C f_X(x\,|\mathbf{C}^r=c,\mathsf{A})f_X(x\,|\mathbf{C}^r=c)f_C(c)\,dc\ .\end{aligned}$$

□

Composability. Even if estimation distribution is easy for a single output, it can be much more difficult for several outputs. We want to get composability results for the upper bounds on posterior probability obtained for a single attacker.

Since $P_C(x,c,\mathsf{A})$ and $P_P(x,\mathsf{A})$ depend on $P_Y(x,y,\mathsf{A})$, and the issue of analyzing several timepoints is integration through all possible values of z, we will only analyze composability of $P_Y(x,y,\mathsf{A})$.

Let $c=(c_1,\ldots,c_n)$ be the true output. First of all, since we want to bound attacker's inference probability by $f_X(x|c)$, we need to take into account at least all elements of c on which $f_X(x|c)$ depends, and we cannot estimate it only based on $f_X(x|c_j)$ for some particular $j\in[n]$. Hence, we need to constrain ourselves to the situations $c=c_a,c_b$, where $f_X(x|c)=f_X(x|c_a)$. E.g. for a counting histogram, when the attacker wants to guess the group a particular individual belongs to, we have $f_X(x|c_1,\ldots,c_n)=\frac{c_j}{m}$, where c_j is the true group to which the input belongs. Indeed, the value m does depend on the other counts as well, but we treat m as a public parameter that makes c_i correlated.

Parallel composition for independent c_a,c_b is similar to the one of DP.

Theorem 3.97. *Let $C=C_a\times C_b$. Let $f_Z(z_a,z_b\,|\mathsf{A})=f_{Z_a}(z_a\,|\mathsf{A})\cdot f_{Z_b}(z_b\,|\mathsf{A})$ for all $z_a\in C_a$, $z_b\in C_b$. We have*

$$P_Y(x,y,\mathsf{A})=\frac{\int_{C_a} f_X(x|z,\mathsf{A})\cdot f_{YC_a}(y,z)f_{Z_a}(z\mid\mathsf{A})\,dz}{\int_{C_a} f_{YC_a}(y,z)f_{Z_a}(z\mid\mathsf{A})\,dz}\ . \tag{43}$$

Proof. For shortness of notation, let us remove the additional knowledge A from all conditional probabilities and make it implicit. This will not affect the proofs, as A is found in the premises of all theorems. By definition of $P_Y(x, y, \mathsf{A})$ (Equation 38), we have

$$\begin{aligned} P_Y(x, y, \mathsf{A}) &= \frac{\int_C f_X(x|z) f_{YC}(y, z) f_Z(z)\, dz}{\int_C f_{YC}(y, z) f_Z(z)\, dz} \\ &= \frac{\int_{C_a}\int_{C_b} f_X(x|z_a) f_Y(y_a, y_b|z_a, z_b) f_Z(z_a, z_b)\, dz_a\, dz_b}{\int_{C_a}\int_{C_b} f_Y(y_a, y_b|z_a, z_b) f_Z(z_a, z_b)\, dz_a\, dz_b} \end{aligned}$$

Assuming that the noise distribution depends only on a single output (as in the case of Laplace noise), we get $f_Y(y_a, y_b|z_a, z_b) = f_{Y_a}(y_a|z_a) \cdot f_{Y_b}(y_b|z_b)$.

For independent variables, $f_Z(z_a, z_b) = f_{Z_a}(z_a) f_{Z_b}(z_b)$. This allows to split both the numerator and the denominator into a product of two independent integrals, and we get

$$\begin{aligned} P_Y(x, y, \mathsf{A}) &= \frac{\int_{C_a}\int_{C_b} f_X(x|z_a) f_{Y_a}(y_a|z_a) f_{Y_b}(y_b|z_b) f_{Z_a}(z_a) f_{Z_b}(z_b)\, dz_a\, dz_b}{\int_{C_a}\int_{C_b} f_{Y_a}(y_a|z_a) f_{Y_b}(y_b|z_b) f_{Z_a}(z_a) f_{Z_b}(z_b)\, dz_a\, dz_b} \\ &= \frac{\int_{C_a} f_X(x|z_a) f_{Y_a}(y_a|z_a) f_{Z_a}(z_a)\, dz_a \cdot \int_{C_b} f_{Y_b}(y_b|z_b)\, f_{Z_b}(z_b)\, dz_b}{\int_{C_a} f_{Y_a}(y_a|z_a) f_{Z_a}(z_a)\, dz_a \cdot \int_{C_b} f_{Y_b}(y_b|z_b)\, f_{Z_b}(z_b)\, dz_b} \\ &= \frac{\int_{C_a} f_X(x|z) f_{YC_a}(y, z) f_{Z_a}(z)\, dz}{\int_{C_a} f_{YC_a}(y, z) f_{Z_a}(z)\, dz}\ . \end{aligned}$$

□

It is more complicated with sequential composition. At another extreme, c_i may be completely correlated, giving $f_{Z_b}(g(z_a)|\mathbf{C}_a^g = z_a) = 1$ for a deterministic function g. In a particular case where $g(z) = z$, we get something similar to sequential composition of DP. This basically allows to use the same z instead of z_a and z_b, so that integration over z_b can be avoided. This can be useful to analyze queries applied multiple times to the same data.

Theorem 3.98. *Let $C = C_a \times C_b$, where $C_a = C_b$. Let $f_{Z_b}(z_a|\mathbf{C}_a^g = z_a, \mathsf{A}) = 1$ for all $z_a \in C_a$. We have*

$$P_Y(x, y, \mathsf{A}) = \frac{\int_{C_a} f_X(x|z, \mathsf{A}) \cdot f_{YC_a}(y_a, y_b, z) f_{Z_a}(z \mid \mathsf{A})\, dz}{\int_{C_a} f_{YC_a}(y_a, y_b, z) f_{Z_a}(z \mid \mathsf{A})\, dz}\ . \tag{44}$$

where $f_{YC_a}(y_a, y_b, z) := f_{YC_a}(y_a, z) \cdot f_{YC_a}(y_b, z)$

Proof. As $f_{Z_b}(z_a|\mathbf{C}_a^g = z_a) = 1$ for all $z_a \in C_a$, we can write $f_Z(z_a, z_b) = f_{Z_a}(z_a)$ and only integrate over C_a. We are left with

$$\begin{aligned} P_Y(x, y, \mathsf{A}) &= \frac{\int_{C_a} f_X(x|z_a) f_Y(y_a, y_b|z_a, z_a) f_Z(z_a)\, dz_a}{\int_{C_a} f_Y(y_a, y_b|z_a, z_a) f_Z(z_a)\, dz_a} \\ &= \frac{\int_{C_a} f_X(x|z, \mathsf{A}) \cdot f_{YC_a}(y_a, y_b, z) f_{Z_a}(z)\, dz}{\int_{C_a} f_{YC_a}(y_a, y_b, z) f_{Z_a}(z)\, dz}\ . \end{aligned}$$

□

Finally, it is possible that the outputs are strongly correlated, but $g(z) \neq z$. Let us prove a small lemma first.

Lemma 3.99. *Let $\mathcal{M}_q$ be any ϵ-differentially private mechanism. Let $z, z' \in q(X)$. Let $R = \max_{x \in q^{-1}(z), x' \in q^{-1}(z')} \|x - x'\|$. For all $y, y' \in \mathcal{M}_q(X)$ we have*

$$f_Y(y|z) \leq f_Y(y|z') \cdot e^{\epsilon R} \ .$$

Proof. We have $z = q(x)$ and $z' = q(x')$ for some $x, x \in X$. Since $q(x)$ can be computed from x, we have $f_Y(y|q(x), x) = f_Y(y|x)$. Let $q^{-1}(z) := \{x \mid q(x) = z\}$. Denote $X_z := q^{-1}(z)$ and $X_{z'} := q^{-1}(z')$. We have

$$\begin{aligned}
\frac{f_Y(y|z)}{f_Y(y|z')} &= \frac{\int_{x \in X_z} f_Y(y|x,z) f_X(x \mid z)\, dx}{\int_{x' \in X_{z'}} f_Y(y|x',z') f_X(x' \mid z')\, dx'} \\
&= \frac{\int_{x \in X_z} f_Y(y|x) f_X(x \mid z)\, dx}{\int_{x' \in X_{z'}} f_Y(y|x') f_X(x' \mid z')\, dx'} \\
&= \int_{x \in X_z} \frac{f_Y(y|x)\ f_X(x \mid z)}{\int_{x' \in X_{z'}} f_Y(y|x') f_X(x' \mid z')\, dx'}\, dx \\
&\leq \int_{x \in X_z} \frac{e^{\epsilon \|x - x'\|}}{\int_{x' \in X_{z'}} f_X(x' \mid z')\, dx'} f_X(x \mid z)\, dx \\
&\leq e^{\epsilon R} \frac{\int_{x \in X_z} f_X(x \mid z)\, dx}{\int_{x' \in X_{z'}} f_X(x' \mid z')\, dx'} = e^{\epsilon R} \ .
\end{aligned}$$

□

For $g(z) \neq z$, the estimation is more difficult, as we want to get a general upper bound for all possible definitions of g. One idea is to apply the worst case bound to the Z_b part.

Theorem 3.100. *Let $C = C_a \times C_b$, $f_{Z_b}(g(z_a)|\mathbf{C}_a^g = z_a, \mathsf{A}) = 1$ for a deterministic function g. We have*

$$P_Y(x, y, \mathsf{A}) \leq \max_{z_a \in Z_a, z_b \in Z_b} \left(\frac{f_{YC_b}(y_b, z_a)}{f_{YC_b}(y_b, z_b)} \right) \cdot \frac{\int_{C_a} f_X(x|z, \mathsf{A}) \cdot f_{YC_a}(y_a, z) f_{Z_a}(z \mid \mathsf{A})\, dz}{\int_{C_a} f_{YC_a}(y_a, z) f_{Z_a}(z \mid \mathsf{A})\, dz} \ .$$

Proof. Let g be a deterministic function such that $f_{Z_b}(g(z_a)|\mathbf{C}_a^g = z_a, \mathsf{A}) = 1$. We have

$$P_Y(x, y, \mathsf{A}) = \frac{\int_{C_a} f_X(x|z_a) \cdot f_{Y_a}(y_a|z_a) f_{Y_b}(y_b|g(z_a)) f_{Z_a}(z_a)\, dz_a}{\int_{C_a} f_{Y_a}(y_a|z_a) f_{Y_b}(y_b|z_b) f_{Z_a}(z_a)\, dz_a} \ .$$

Differently from the previous case, we cannot compute the quantity more precisely unless we know g. We rewrite the expression as

$$P_Y(x, y, \mathsf{A}) = \int_{C_a} f_X(x|z) \cdot f_{Y_a}(y_a|z) \cdot f_{Z_a}(z) \cdot \frac{1}{\int_{C_a} f_{Y_a}(y_a|z_a) \frac{f_{Y_b}(y_b|z_b)}{f_{Y_b}(y_b|g(z))} f_{Z_a}(z_a)\, dz_a}\, dz \ .$$

A trivial upper bound on $\frac{f_{Y_b}(y_b|g(z))}{f_{Y_b}(y_b|z)}$ is $\max_{z_a \in Z_a, z_b \in Z_b} \frac{f_{Y_b}(y_b|z_a)}{f_{Y_b}(y_b|z_b)}$. This upper bound is independent of the integration variables and can be taken out of integration. □

We can instantiate Theorem 3.100 on an ϵ-DP mechanism.

Corollary 3.101. *Let $C = C_1 \times \cdots \times C_n$. Let an ϵ-DP mechanism w.r.t. norm $\|\cdot\|$ be applied to each C_j. Let $\max_{x,x' \in X} \|x - x'\| = m$. We have*

$$P_Y(x, y, \mathsf{A}) \leq e^{\epsilon \cdot m(n-1)} \cdot \frac{\int_{C_1} f_X(x|z, \mathsf{A}) \cdot f_{YC_1}(y_1, z) f_{Z_1}(z \mid \mathsf{A})\, dz}{\int_{C_1} f_{YC_1}(y_a, z) f_{Z_1}(z \mid \mathsf{A})\, dz} \ .$$

constr ::= `exact`
| `total` *int*
| `set` {*string*}
| `range` *double double*

| `totalUnif` *int*
| `setUnif` {*string*}
| `rangeUnif` *double double*

| `setPrior` {(*string* , *float*)}
| `rangePrior` *float* {(*float* , *float*)}

| `normal` *double double*

Figure 44: Possible Constraints on Table Attributes

Proof. By assumption of Corollary 3.101, we are dealing with an ϵ-DP mechanism. by Lemma 3.99, we have $\frac{f_{Y_b}(y_b|g(z))}{f_{Y_b}(y_b|z)} \leq e^{\epsilon} \cdot \max_{x,x'} \|x - x'\| \leq e^{\epsilon} \cdot m(n-1)$. The claim now follows directly from Theorem 3.100. □

3.3.11 Propagation of Constraints on Attributes. In both sensitivity and guessing advantage analyses, we make use of the bounding ranges from which the values may come. While the upper bound on derivative sensitivity may be infinite for $X = \mathbb{R}$, it can be much smaller for a bounded subset $X \subset \mathbb{R}$. The user had to provide possible bounding ranges on the inputs himself. For small models, such approach is sufficient. However, in large models, one might want to compute sensitivity and guessing advantage w.r.t. some intermediate table, e.g. a receiving party may want to know how much leaks about the data that has already gone through some preprocessing. Knowing ranges of intermediate tables can also be useful for composition. While the user can set bounding ranges only on the direct input tables, we need automated constraint derivation to determine the ranges of attributes of intermediate tables as well.

The list of possible constrains on table attributes is given in Figure 44. We see that the constraints can be split to two main types: discrete and numeric.

We consider SQL queries of the form

`SELECT a1 AS b1,...,ak AS bk FROM t1 AS s1,...,tn AS sn WHERE` *condition* `GROUP BY a1,...,ak-1 .`

3.3.11.1 Numeric Attributes. The numeric constraints of Figure 44 allow to define just an interval $[a, b]$ for some $a \leq b \in \mathbb{R}$, as well as some distribution on that interval, which is currently just a concatenation of a finite number of uniformly distributed intervals. Propagation of distributions is much more complicated than propagating numbers. In our analyses, defining a range without a particular distribution gives a safe noise overestimation, so first of all we may concentrate on propagating ranges without maintaining the distribution.

First of all, we discuss how to compute output ranges for real functions $f : \mathbb{R} \to \mathbb{R}$. These would correspond to the computation under `SELECT` statement. In general, overestimated ranges give us higher function sensitivity and result in larger noise, so it is safe to make an overestimation of the actual range.

Interval arithmetic. An obvious solution to function output range estimation is to use interval arithmetic. We need to define operations between intervals. For example some basic arithmetic operations might be defined as

$$\begin{aligned}
[a,b] + [c,d] &= [a+b, c+d] \\
-[a,b] &= [-b,-a] \\
[a,b] \times [c,d] &= [\min\{ac,ad,bc,bd\}, \max\{ac,ad,bc,bd\}]
\end{aligned}$$

These operations can then be composed to find the output range for more complex expressions. Since not all such functions yield the exact output range we can divide the input interval into smaller subintervals to produce more precise results, as described e.g. in [63, p. 306].

When using interval arithmetic to evaluate functions where the same variable appears multiple times, it tends to give us very wide output intervals to the point where the result is completely useless. This happens because we assume that all the variables in the function are independent [64, p. 36]. Let us look for alternative methods that solve this problem.

Affine arithmetic. An alternative to interval arithmetic is the *affine arithmetic*. Using affine arithmetic we can write the intervals in the affine form

$$\hat{x} = x_0 + \epsilon_1 x_1 + \epsilon_2 x_2 + \ldots + \epsilon_n x_n$$

where $\epsilon_i \in [-1,1],\ i \in [1 \ldots n]$. For example, if $n = 1$, then x_0 can be viewed as the central point of the interval, and sliding $\epsilon_i \in [-1,1]$ gives the values $[x_0 - x_1, x_0 + x_1]$. If two affine representations use the same ϵ_i, it means that these two variables are correlated.

Let us look at the function $f(x) = x - x$. Assuming $x \in [0,3]$, we can find the output range of the function using interval arithmetic:

$$\begin{aligned}
[0,3] - [0,3] &= [0,3] + (-[0,3]) \\
&= [0,3] + [-3,0] \\
&= [-3,3]
\end{aligned}$$

It turns out that interval arithmetic does not give us the exact answer $[0,0]$, but instead gives us a much wider interval. Now let us find the output range using affine arithmetic:

$$\begin{aligned}
(1.5 + 0.5\epsilon_1) - (1.5 + 0.5\epsilon_1) &= (1.5 - 1.5) + (0.5 - 0.5)\epsilon_1 \\
&= 0 + 0\epsilon_1 = 0
\end{aligned}$$

We get the exact result because every occurrence of variable x has the term $x_1\epsilon_1$, which cancel out after the subtraction. If we wanted to write different variables instead we would write

$$\begin{aligned}
\hat{x} &= x_0 + x_1\epsilon_1 \\
\hat{y} &= y_0 + y_1\epsilon_2
\end{aligned}$$

Converting between intervals and affine forms. We can convert any affine form $\hat{x} = x_0 + x_1\epsilon_1 + \ldots + x_n\epsilon_n$ to an interval as follows:

$$x = [x_0 - r, x_0 + r],\ \textit{where } r = \sum_{i=1}^{n} |x_i|.$$

It should however be kept in mind that by converting to interval form, we lose the information about the dependencies between variables. We can also get the affine form of the interval $[a,b]$

$$\hat{x} = \frac{a+b}{2} + \frac{b-a}{2}\epsilon_1 \ ,$$

as described in [64, p.47-49].

When doing this conversion, each different variable should have its own epsilon symbol. Libaffa, a C++ implementation of affine arithmetic uses a map data structure for mapping between variable names and corresponding coefficients [65]. In Haskell it can be defined like this:

```
data AAF a = AAF a (Map String a) a
```

In this definition the first field represents the 0-th term. The map binds variable names with corresponding terms. All non-affine operations such as multiplication and square root add new terms after each operation (i.e. error terms) and to keep the code simple, we collect all these terms into a single error term, which is what the last field in the AAF structure is for.

Non-affine functions. Some functions do not give us an exact affine answer. For example consider multiplying two affine forms $\hat{x} = x_0 + x_1\epsilon_1 + \ldots + x_n\epsilon_n$ and $\hat{y} = y_0 + y_1\epsilon_1 + \ldots + y_n\epsilon_n$. We would end up with an affine part

$$\hat{z}^a = x_0y_0 + (x_0y_1 + y_0x_1)\epsilon_1 + \ldots + (x_0y_n + y_0x_n)\epsilon_n$$

and a non-affine part

$$z^* = (x_1\epsilon_1 + \ldots + x_n\epsilon_n)(y_1\epsilon_1 + \ldots + y_n\epsilon_n)$$

Since we want an affine approximation of the result, we have to add a new term to the affine part to approximate the non-affine part

$$\hat{z} = \hat{z}^a + (x_1 + \ldots + x_n)(y_1 + \ldots + y_n)\epsilon_{n+1} \ ,$$

as described in [64, p.70].

In practice it makes sense to collect all such terms of subsequent operations into a single term as we did in the Haskell implementation example. This makes sure that the affine forms do not grow unnecessarily large. It can even be reasonable to have more than one error term to reduce error when multiplying or raising the affine forms to some power [64, p. 81].

Min-range approximation. We can generalize non-affine functions in a more general way. Let us consider some binary non-affine function $f(\hat{x}, \hat{y})$. We can approximate this function by finding some good values for the coefficients $\alpha, \beta, \zeta \in \mathbb{R}$ in

$$f^a = \alpha\hat{x} + \beta\hat{y} + \zeta$$

Computationally one of the best methods is to use the min-range approximation. We find the coefficients in such a way that the resulting range of the function is as narrow as possible. An example of min-range approximation of a log function is given in Figure 45, where the input range $[a, b]$ gives an output range $[c, d]$, and all function values stay inside the blue skewed rectangle. Even though this method is not optimal in terms of minimizing the error, it is easier to implement and has a smaller overshoot than Chebyshev approximation, which aims to minimize the maximum error [64, p. 56,64-65].

Fixing the values of variables. Let us look at a case where we need to find the output range of a function when we know the range of one variable, and a set of exact values for a second variable. We have affine forms $\hat{x} = x_0 + x_1\epsilon_1$ and $\hat{y} = y_0 + y_2\epsilon_2$ representing the respective variables. Once we evaluate the function using these affine forms, we end up with a new affine form

$$\hat{z} = z_0 + z_1\epsilon_1 + z_2\epsilon_2 + z_3\epsilon_3 \ .$$

The noise symbols ϵ_1 and ϵ_2 correspond to the variables x and y respectively. If we later want to fix y to some value a we can substitute $\epsilon_2 = \frac{a}{y}$ which gives us

$$\hat{z}' = z_0 + z_1\epsilon_1 + z_2\frac{a}{y} + z_3\epsilon_3 \ .$$

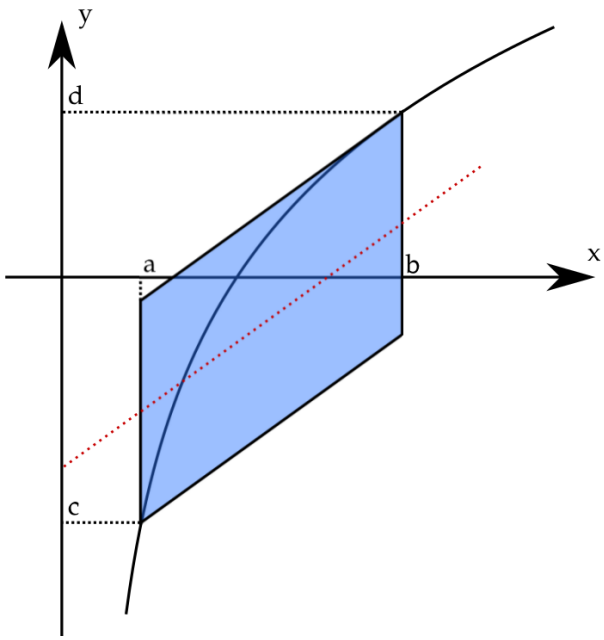


Figure 45: Min-Range Approximation for the log **Function**

This means we only have to find $\hat{z}$ once and then do a relatively cheap computation to fix the values afterwards. It is also possible to fix the value to some subrange of $[y_0 - y_2, y_0 + y_2]$. This approach is very useful when we have a large number of similar computations, like applying the same function to n rows of an input table. It can also be applied in cases where a variable is represented as a discrete set of possible values rather than a range.

Subdividing input ranges. When estimating the output range of an expression given some input range, it is possible to get more accurate results by dividing the input range into smaller subdivisions. Each of these subdivisions can then be evaluated separately which results in smaller zonotopes that follow the function more closely [63, p. 306-307]. An example of dividing input range $[a, b]$ into smaller subranges for a log function is given in Figure 46. We see that the function values are now covered by the blue skewed rectangles much more tightly. When we later want to fix the input to some specific value or range, we only have to do that in the input subdivisions that contain the new value.

Distributions with unbounded domain. In practice, some data attributes may not have nice upper and lower bounds. One possible realistic distribution is the normal distribution $N(\mu, \sigma^2)$. Since the range of normally distributed inputs is unbounded, it seems that knowing that the data is distributed normally will not provide good bounds on the function output. However, the values still remain within a certain range with high probability. If the distribution has bell shape, it does not make sense to consider elements that are too far from the center. For $N(\mu, \sigma^2)$ we can take $R = \mu + 3 \cdot \sqrt{2}\sigma$, which covers $\mathsf{erf}(3) \approx 0.9999779$ of the input space. We can then assume that our inputs are coming from the range $[\mu - R, \mu + R]$. In general, we can let η be a parameter and find ξ such that $\mathsf{erf}(\xi) = 1 - 2^{-\eta}$. E.g. we have $\xi \approx 5.1$ for $\eta = 40$.

This approach can be generalized to any distribution with well-defined cumulative density function (CDF) $F(x)$. The probability weight of the range $[\mu - R, \mu + R]$ can be found by definition of CDF as $F(\mu + R) - F(\mu - R)$.

3.3.11.2 Filters and Groups. So far, we have shown how to compute the ranges of values computed under the SELECT statement, and how to optimize the computation of range w.r.t. the number of rows in the input table. We now want to learn how these ranges change after some grouping and/or filtering is

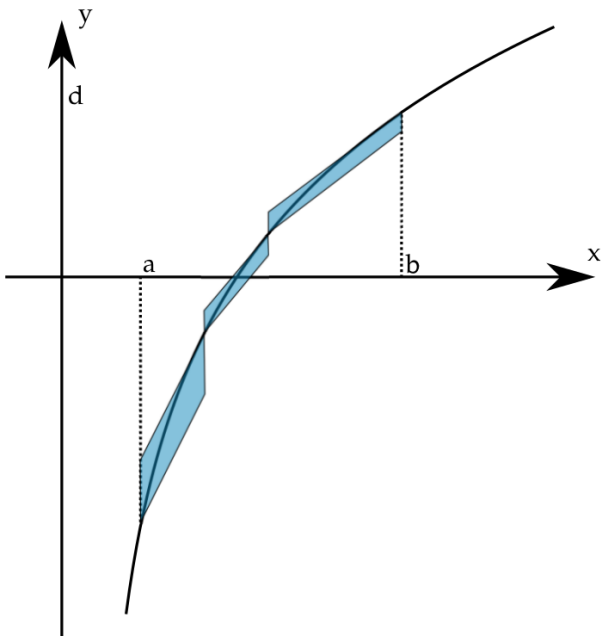


Figure 46: Subdividing the Input Range Gives More Accurate Results

applied. Similarly to the sensitivity analysis, we consider a finite number m of possible groups, so that a single GROUP BY query can be viewed as m instances of query with an additional filter that selects the group.

It remains to show how to handle the filters. It is easy to apply a public filter, as we can discard the mismatching row directly. It is more difficult with a filter that contains private values, for which we may only use ranges. Similarly to sensitivity analysis, a filter can be viewed as a function $\sigma(x) \in \{0, 1\}$, and if we know that an aggregation is applied afterwards, we can leave e.g. $\sigma(x)$ for COUNT queries and $\sigma(x) \cdot f(x)$ for SUM queries. It is more complicated for queries without aggregations, as some ranges may allow for both positive and negative filtering, so we even do not know whether the row belongs to the result or not. There are three main possible ways to go:

1. Allow leaking additional information and use exact value to determine whether to leave the row.
2. Leave only those rows that definitely satisfy the filter.
3. Remove only those rows that definitely do not satisfy the filter.

In general, in derivative sensitivity analysis we expect that the number of rows is public, so if we use the resulting table in sensitivity or guessing advantage analysis directly, then we will do it on assumption that the attacker has learned the total number of rows, which is fine, as we assume that the DP attacker is given "all the records except the one he is trying to guess" anyway.

The other two approaches would give an attacker a lower and an upper bound on the number of rows in the table. This range would have a potential application to combined sensitivity analysis, but we have not considered it so far. If the output is not used in the analysis immediately, but is applied in some subsequent query, then the third approach leaves the greatest variety of possible rows and gives a valid range of possible values, so this is what we need for computing overestimated ranges. If the following query is an aggregation, then for all these rows we will have $\sigma(x) \in \{0, 1\}$, which can be treated as an ordinary interval.

We let the analyzer decide which approach is the most suitable. This depends on the type of the analyzed query.

- **Plain SELECT-query**. The range of an output attribute is defined as the union of all intervals. Hence, a safe overapproximation would be to leave all rows for which we are not sure whether they will be filtered out or not. We apply the method (2).

- **Aggregation query**. The range of an output attribute depends on the aggregation type.
  - **COUNT**. We get the smallest possible count c_{lb} by leaving only those rows that definitely pass the filter, i.e. method (3). We get the largest possible count c_{ub} by leaving only those rows that definitely pass the filter, i.e. method (2). We define the range of a COUNT query output as $[c_{lb}, c_{ub}]$.
  - **SUM**. If we know in advance that the summed values are non-negative, then we can find the sum range similarly to the COUNT, computing the sum s_{lb} over lower bounds of ranges of the rows left by method (3), and s_{ub} over upper bounds of ranges of the rows left by method (2). If the summed values can be negative, we need to include *all* negative lower bounds into s_{lb}, and *all* positive upper bounds into s_{ub}. Finally, we define the range of a SUM query output as $[s_{lb}, s_{ub}]$.
  - **MIN/MAX**. For MIN, we compute the minimum m_{lb} over lower bounds of ranges of the rows left by (2), and the minimum m_{ub} over lower bounds of ranges of the rows left by (3).For MAX, we compute the maximum m_{lb} over upper bounds of ranges of the rows left by (3), and the maximum m_{ub} over upper bounds of ranges of the rows left by (2). We define the range of a MIN/MAX query output as $[m_{lb}, m_{ub}]$.

3.3.11.3 Discrete Attributes. So far, our analyzers are using discrete attributes only for comparison. one of the following may happen to a discrete attribute x:

- copied to the output table as it is (in plain SELECT-queries, or when grouping by x);
- used in a comparison-based filter.

In these cases, a discrete attribute is handled similarly to a numeric attribute. Instead of a *range*, we define a data structure *set*, which contains a (possibly overestimated) set of values S from which this attribute may come. We allow a *universal set*, which is analogous to the interval $[-\infty, \infty]$, and an *empty set*.

When a row is being filtered, we check whether it is possible to pass the filter given the ranges of attributes that we have. We mark whether it *always* passes the filter, *never* passes the filter, or *both outcomes are possible*. For each attribute, we constrain the output range based on the filter, possibly leaving an overapproximation if it cannot be easily extracted. E.g., if a filter involves a combination of different attributes that are not separated by AND, e.g. $x + y \leq 5$, then this dependency is not included into the resulting constraints.

If the row contains any discrete attributes, the analysis is analogous. For each $s \in S$, we check whether s passes the filter or not. If s is compared with a constant, this check is trivial. If s is compared to another attribute $s' \in S'$, then we can set output ranges of both attributes to $S \cap S'$. Similarly to numeric attributes, dependency of different attributes is not included into the resulting constraints.

3.3.12 Differential Privacy for Time Series. So far, we have studied the case where a query is executed only once. Even if we have enough budget to execute more than one query, those queries are executed on the same database, which does not change in the meantime. Now we consider the case where the database changes over time and we execute the same query at several points in time. Thus we will get a time series. We would like to release the result of the query at each of the time points in a way that all the released values together are still differentially private w.r.t. all the inputs (over all time points).

A naive approach of obtaining differential privacy for time series is to simply split the privacy budget ϵ between the n time points, releasing the query result at each time point $\frac{\epsilon}{n}$-differentially privately. This makes the noise n times larger than when executing the query only once. In practice, the input database usually does not change very fast over time, thus the query results at different time points should also be correlated. Then we can use an approach that is used outside the differential privacy world to smoothe the noisy time series of measurements obtained from sensors, namely *Kalman filters*, which we consider

in Sec. 3.3.12.1. We also obtain a new method for generating the differential privacy noise using smooth derivative sensitivity, where we get to know the exact noise level of the released value.

Note that in the current section, we often use the phrase "local/derivative sensitivity" because most of the discussed methods apply to both local sensitivity (where the distance over inputs is integer-valued) and derivative sensitivity (where the distance over inputs is real-valued). Even when we use the phrase "derivative sensitivity", it is often applicable to local sensitivity as well.

Similarly to Sec. 3.3.9, because in this section we will use two different kinds of derivative sensitivity (w.r.t. row multiplicities and w.r.t. components), we call the derivative sensitivity w.r.t. components (i.e. in Banach spaces), *Banach sensitivity* or *Banach derivative sensitivity*.

3.3.12.1 Kalman Filters. Differential privacy can be achieved for time series using global sensitivity. This has been done in [66] using a Kalman filter. In this section, we discuss how to use Kalman filters with local or derivative sensitivity. We also obtain a new method for generating the differential privacy noise using smooth derivative sensitivity. It can be used with a slightly smaller budget than the old method and we get to know the exact noise level of the released value. If the budget is not very small and we do not need to learn the exact noise level, the old method still gives about e times better accuracy.

Kalman filters. Kalman filters can be used to take a series of noisy measurements at different time points (e.g. GPS data) and make it less noisy. To produce an estimate at the current time point, we use not only the noisy measurement at the current time point but also those at previous time points. If the underlying value does not change too fast then the previous measurements are additional estimates of the current value, although less precise. Thus we have several (at least partially) independent estimates of the current value, with different precision. Then we take their weighted average (with more precise estimates having higher weight) to get an estimate that is more precise than any of the initial estimates.

Kalman filters work in real time, i.e. the current estimate does not depend on future measurements. Also they can be implemented in constant memory, i.e. it is not necessary to save all the previous measurements. Also, execution time per measurement does not increase with the number of measurements.

Kalman filters for differential privacy. Kalman filters seem suitable for differential privacy because differentially private queries produce noisy estimates of the query result, similarly to noisy measurements from sensors. The difference is that measurements from sensors usually have Gaussian noise but differential privacy uses Laplace noise or generalized Cauchy noise (although Gaussian noise is also possible at the expense of higher noise levels and achieving only (ϵ, δ)-DP instead of ϵ-DP).

To use a Kalman filter, we need to estimate how much the underlying value has changed from the previous time point. This is assumed to be normally distributed. We also need to estimate the noise level of the measurement. When global sensitivity is used for differential privacy, this noise level is constant and public. With local/derivative sensitivity, we need to take extra steps to determine it, which we will discuss later.

The two sources of error must be independent, and also independent of these in previous time periods. The noise levels are used to compute the weights of different measurements in computing the more accurate estimate. If the noise level estimates are incorrect or the noises are not independent then it is not guaranteed anymore that the estimate is more accurate than the measurements. Differential privacy is still guaranteed because transformations do not weaken differential privacy.

Revealing the noise level for derivative sensitivity. When global sensitivity is used for differential privacy, this noise level is constant and public. With local/derivative sensitivity, we need to take extra steps to determine it.

One way is to use a part of the privacy budget to reveal the β-smooth derivative (or local) sensitivity differentially privately. β-smoothness guarantees that the logarithm of this sensitivity has global sensitivity at most β. Thus we can use global sensitivity to reveal this logarithm differentially privately. The

problem with this approach is that we only reveal an approximation of the noise level used for revealing the query result, not the exact noise level, thus the Kalman filter will be less accurate.

Another way is to first use a part of the privacy budget to reveal a noise level differentially privately (actually we reveal the logarithm of the noise level as in the previous approach, to avoid the chance of the noise level becoming negative after adding noise) and then generate Laplace noise with exactly this noise level to reveal the query result. Then we can use the exact noise level in the Kalman filter. The problem is how to find this appropriate noise level. If we reveal the logarithm of the actual β-smooth sensitivity then the added noise may be negative and the revealed noise level thus too low to achieve differential privacy for the query result.

Instead, we add another term to the logarithm, so that after adding noise, the revealed noise level will be with probability $1 - \delta$ enough to achieve ϵ-differential privacy for the query result. Since this δ does not depend on the input distribution, we will get (ϵ, δ)-differential privacy. The revealed noise level is

$$M \cdot e^{n \cdot k + Lap(k)}$$

where

- $n \cdot k$ is the added term to make $n \cdot k + Lap(k)$ non-negative with probability $1 - \delta$;
- $M = c/b$ is the computed noise level;
- c is the β-smooth sensitivity;
- $\delta = \frac{1}{2} \cdot e^{-n}$;
- $k = \beta/a$;
- $\epsilon = a + b$;

The privacy budget ϵ is split into a and b, where a is used for revealing the noise level and b is used for revealing the query result using this noise level.

The median noise level depends on how the budget is split. Let us find the value of a for which the median noise level is minimal. The logarithm of the median noise level is

$$\frac{n\beta}{a} - \ln(\epsilon - a) + \ln c$$

The minimum is achieved when the derivative w.r.t. a is 0, i.e.

$$-\frac{n\beta}{a^2} + \frac{1}{\epsilon - a} = 0$$

$$\epsilon - a = \frac{a^2}{n\beta}$$

$$a^2 + an\beta - \epsilon n\beta = 0$$

$$(a + n\beta/2)^2 = n^2\beta^2/4 + \epsilon n\beta$$

$$a = -n\beta/2 + \sqrt{n^2\beta^2/4 + \epsilon n\beta}$$

We can try this out in Haskell:

```
let f nb eps = let a = -nb/2 + sqrt(nb^2/4 + eps*nb); b = eps-a in (exp(nb/a)/b, a)
let g nb eps = (1/(eps-nb), nb)
let h eps nb = (f nb eps, g nb eps, fst(f nb eps)/fst(g nb eps))
Prelude> h 1 0.01
((1.2276273246518064,9.512492197250393e-2),(1.0101010101010102,1.0e-2),1.2153510514052883)
Prelude> h 1 0.1
((1.9839324650212786,0.27015621187164246),(1.1111111111111112,0.1),1.7855392185191508)
Prelude> h 1 0.4
((4.418001576289371,0.46332495807108004),(1.6666666666666667,0.4),2.650800945773623)
Prelude> h 1 0.5
```

```
((5.43656365691809,0.5),(2.0,0.5),2.718281828459045)
Prelude> h 1 0.6
((6.600169279739,0.5306623862918074),(2.5,0.6),2.6400677118956)
Prelude> h 1 0.9
((11.20422267584516,0.6000000000000001),(10.000000000000002,0.9),1.120422267584516)
Prelude> h 1 0.95
((12.170092683730568,0.6092624221100721),(19.999999999999982,0.95),0.6085046341865289)
Prelude> h 1 1
((13.203179065212284,0.6180339887498949),(Infinity,1.0),0.0)
Prelude> h 1 2
((57.34058739617453,0.7320508075688772),(-1.0,2.0),-57.34058739617453)
Prelude> h 1 3
((212.31841055007064,0.7912878474779199),(-0.5,3.0),-424.6368211001413)
Prelude> h 1 4
((728.6360040835475,0.8284271247461903),(-0.3333333333333333,4.0),-2185.9080122506425)
Prelude> h 1 10
((656034.405125172,0.9160797830996161),(-0.1111111111111111,10.0),-5904309.646126548)
Prelude> h 1 20
((2.7664644276074154e10,0.9544511501033224),(-5.263157894736842e-2,20.0),-5.2562824124540894e11)
```

We see that the overhead of revealing the noise level is largest when $n\beta = \epsilon/2$, then the noise increases e times if the total budget is not increased compared to the case where noise level is not revealed. If $n\beta$ approaches ϵ then the new method even starts to use less noise than the previous method, even though it releases more information. If $n\beta \geq \epsilon$ then the previous method would require infinite noise level and cannot be used at all. The new method can be used but as $\frac{n\beta}{\epsilon}$ exceeds 1, the noise level starts to increase exponentially in $\frac{n\beta}{\epsilon}$ and soon becomes impractical. Nevertheless, we may be able to use this method practically with a few times smaller budget than the old method. This is possible because the new method uses Laplace noise whose level is determined by an exponent of another Laplace noise. This combined noise distribution has a bit heavier tails than the ordinary Laplace distribution.

3.3.12.2 Local/Derivative vs Global Sensitivity in Kalman Filters. Because Kalman filters require splitting the budget between the released values and local/derivative sensitivity cannot be practically used with very small budgets per released value, we discuss why we need local/derivative sensitivity at all. We propose a modified version of global sensitivity that often has a reasonable magnitude when the actual global sensitivity is infinite or very large. This can be used in the simpler methods of achieving DP using global sensitivity. We will see that there are still cases where derivative sensitivity is needed.

We may question why we want to use local/derivative sensitivity instead of the more standard global sensitivity. The main reason is because global sensitivity may be infinite or very large. However, we can often use something similar to global sensitivity that has a more reasonable magnitude and can still be used with the simpler methods that achieve differential privacy using global sensitivity, with a slight loss in privacy. We try to find an example where even this modified global sensitivity is not good enough and we really have to use the more complicated methods of local/derivative sensitivity.

$(1-\delta)$-global sensitivity. If the actual global sensitivity is infinite or very large then we may instead use a value that is with probability at least $(1-\delta)$ an upper bound on the local (or derivative) sensitivity. Then we get ϵ-differential privacy with probability at least $(1-\delta)$. This is similar to but not the same as (ϵ, δ)-differential privacy because it depends on the actual input distribution but (ϵ, δ)-differential privacy holds for any input distribution.

Derivative sensitivity vs $(1-\delta)$-global sensitivity. If the actual global sensitivity is infinite or very large then we have a choice between using either derivative sensitivity or $(1-\delta)$-global sensitivity. With derivative sensitivity we get classical (ϵ, δ)-differential privacy, with $(1-\delta)$-global sensitivity we get something similar but not the same.

To use $(1-\delta)$-global sensitivity, we need to know something about the input distribution. If we do not know the exact input distribution then at least we need to find some values δ and c such that with probability at least $1-\delta$, the input is such that the derivative sensitivity at that input point is at most c.

To use derivative sensitivity, we do not need to know anything about the input distribution. Instead, we need to use a value of β and find a β-smooth upper bound on the derivative sensitivity. The noise level depends on β and on the actual input, choosing the best β can be difficult.

Also, derivative sensitivity requires a higher noise level for the same sensitivity than global or $(1-\delta)$-global sensitivity. It is about 2 times higher if we do not need to reveal the noise level and $2e$ times higher if we do. Thus we need the median (over an input distribution) β-smooth derivative sensitivity to be at least $2e$ times smaller than its $(1-\delta)$-quantile for derivative sensitivity to be advantageous over $(1-\delta)$-global sensitivity.

Thus the derivative sensitivity must vary over a quite large range over the possible inputs. If the query uses only one table and uses it only once and the norm used for combining row norms is ℓ_1 then there are some inputs from which the large range of sensitivity can be achieved by changing only one row. Thus β is high and derivative sensitivity cannot be practically used with small budgets.

So let us use the table twice:

```
SELECT count(*) FROM t as t1, t as t2 WHERE t1.groupId = t2.groupId
```

Let each row belong to a different individual and each individual belong to one group, denoted by the field `groupId` in the row. If an individual of row i belongs to a group with m members then the derivative sensitivity w.r.t. row i is $2m$ and is $\frac{1}{m}$-smooth at this point. The total sensitivity of the query is determined by the largest groups thus we can make β quite small, as long as the largest groups are big enough with high probability.

With a small β, we can use a Kalman filter with derivative sensitivity (using the method from Sec. 3.3.12.1 that also reveals the noise level). The smaller the β, the larger the number of time points we can reveal before the budget is exhausted. If we need to reveal an even longer time series (e.g. an infinite time series) then the budget needs to regenerate over time. If β is smaller then it does not need to regenerate as fast as for a larger β.

3.3.12.3 An Alternative to Kalman Filter. Instead of releasing the query result at each time point, we may release the *change* in the query result from the previous time point (in addition to the query result at the first time point of the series). The changes in non-overlapping time periods are likely to be less correlated than the query results at the end of these time periods. Thus each released change does not leak too much about other released changes, which allows reusing some of the budget.

In this section, we assume that the changes are completely independent, thus the whole budget can be reused at each time point. There is still some overhead over the single execution because we are interested in the query results at different time points and each of these must be computed from several released change values. This overhead can be made polylogarithmic, which is much better than the $\Omega(\sqrt{n})$ overhead of the Kalman filter. The disadvantage is that this approach assumes that the changes in non-overlapping time periods are independent (i.e. the sets of input rows that they depend on are disjoint, or if we need user-level privacy instead of event-level, then the sets of users on whose inputs they depend on must be disjoint). Otherwise the privacy is not guaranteed, although accuracy is. On the other hand, with Kalman filters, privacy is guaranteed but accuracy depends on the input distribution. As privacy is usually more important than accuracy, we can use the Kalman filter in the general case but there is a class of queries (considered in the following sections) for which our approach works as well.

We use an algorithm similar to one in [67]. The difference is that they release the results of the query at each time point while we release the changes in the query result over certain time periods. The noises they add to each released value, are, however, not independent and are generated in roughly the same way as in our algorithm, which allows keeping the overhead polylogarithmic is both their and our algorithm. Their algorithm also achieves *pan-privacy*, a stronger version of differential privacy where the attacker can not only see the released outputs but can also look at the internal state of the mechanism

at one point in time that the attacker chooses. Our algorithm can also be modified to achieve pan-privacy, at the expense of increasing the noise level $\sqrt{2}$ times.

Suppose that changes in the query result during non-overlapping time periods are independent and normally distributed. This is also assumed for Kalman filter. Kalman filter reveals the whole query result at each time point. Those values are not independent and thus we have to split the privacy budget between them, leaving a very small budget for each time point.

Suppose that we instead reveal the change in the query result between every two successive time points, in addition to the query result at the first time point. Those revealed values are mutually independent. Then it is possible that making a same change of same magnitude in the change of query result at many time points is not more important than making it in only one time point, i.e. we could use the ℓ_∞ norm instead of the ℓ_1 norm. Then we can reuse the same budget for each time period that does not overlap with the time periods for which budget has already been used. Note that the ℓ_∞ may not always be allowed and then we still need to use the Kalman filter.

Suppose that it is allowed and we use the simplest method of using the whole budget ϵ to reveal the change in query result between each two successive time points. Let those changes be $x_1, x_2, \ldots, x_n$ and let the query result at the first time point be r_0. Then we can reconstruct the time series as $r_i = r_0 + \sum_{j=1}^{i} x_j$ where r_i is the query result at time point i. The noise level of the elements will be $O(\sqrt{n})$.

We can do better. Instead of using the whole budget ϵ to reveal each x_i, we use only ϵ/s for this (where $s = \log_2 n$), then use ϵ/s to reveal each $x_{2i-1} + x_{2i}$, the same amount to reveal each $x_{4i-3} + x_{4i-2} + x_{4i-1} + x_{4i}$, and so on. I.e. for each $k = 0, 1 \ldots, s-1$, we use ϵ/s to reveal each $\sum_{j=1}^{2^k} x_{2^k(i-1)+j}$. Then each r_i can be reconstructed as a sum of r_0 and m revealed values corresponding to changes in periods whose lengths are different powers of two, and m is the number of bits that are 1 in the binary representation of i. The noise level of the r_i will be $O(\sqrt{\log_2 n})$.

If we use derivative sensitivity then using only $\epsilon/\log_2 n$ for each query may be a problem because the noise level starts to increase very fast once the budget goes below a certain value. The old method of using derivative sensitivity can even not be used at all. Thus we generalize the current method to use a-ary representation instead of binary. Then we need $\epsilon/\log_a n$ budget for each query while the noise level increases to $O(\sqrt{(a-1)\log_a n})$.

If the output is a linear function of input and we use the ordinary absolute value norm for each row (this norm is separate from the norm used to combine rows) then the derivative sensitivity is constant and has no advantage over global sensitivity. To make it have an advantage, we may instead use the square root norm (changing the square root of an input row by 1 corresponds to distance 1).

So far we assumed that changes in the query result during non-overlapping time periods are independent and normally distributed. We can actually remove this restriction if we are satisfied with event-level privacy instead of user-level privacy. Then each row corresponds to an event. In the following sections, we show how to achieve differential privacy on event level.

3.3.12.4 Sensitivity w.r.t. Changes Inside Rows. In this section, we show how DP can be achieved using derivative sensitivity w.r.t. component (Sec. 3.3.8.2), i.e. *Banach sensitivity*. Note that, while Banach sensitivity requires that the number of rows in each table is fixed, we still allow to add and remove rows from timepoint to timepoint, but we only treat the content of the rows as private, and and do not attempt to conceal the fact that a row has been added or removed. We will discuss how to do the latter in Sec. 3.3.12.5.

Queries with a single table. We start from a simple case where the query applies a function on each event and sums the results of the functions from all events. The result contributed by each event must not depend on other events. Thus the class of queries that we can compute is the SQL SUM queries on a single table without self-joins:

$$\texttt{SELECT SUM}(f(a_1, \ldots, a_n))\ \texttt{FROM}\ t$$

We can also allow filters because these can be embedded in the function f (multiplying the original f by a function that returns 1 for rows satisfying the filter and 0 for other rows):

$$\texttt{SELECT SUM}(f(a_1, \ldots, a_n))\ \texttt{FROM}\ t\ \texttt{WHERE}\ p(a_1, \ldots, a_n)$$

where a_i are the attributes (columns) of table t. In this section, we will further assume that any filter is already included in the function f.

At each time point, the query is evaluated on those rows that are already in the table, i.e. those events that have already occurred. Over time, rows can only be added to the table, never removed or changed.

Note that the times when rows arrive are not private, only the content of the rows is. This is not a problem if the rows arrive at regular intervals, but if the intervals are irregular then they are leaked.

Let us first compute the sensitivity w.r.t. a row $(a_1, \ldots, a_s)$ added at time point i. We compute a β-smooth upper bound on $\mathrm{DS}_f(a_1, \ldots, a_s)$. Let it be c_i. We also compute the change at time point i, which is $x_i = f(a_1, \ldots, a_s)$. Let the total number of time points be n. Then we can use $\epsilon_1 = \frac{\epsilon}{\log_2 n}$ for revealing x_i. The noise level L_i will then be computed according to the values of ϵ_1, c_i, β, and the noise distribution that we want to use (generalized Cauchy with parameter γ achieving ϵ-DP, Laplace with $\delta_1 = \frac{\delta}{e^\epsilon \log_2 n}$ achieving (ϵ, δ)-DP). Then we generate noise N_i with level L_i, add it to x_i, and reveal the sum $x_i + N_i$.

In addition to revealing the noisy versions of single changes x_i, we also need to reveal some sums of changes in consecutive time periods, e.g. $x_{2i-1} + x_{2i}$ or $x_{4i-3} + x_{4i-2} + x_{4i-1} + x_{4i}$.

For all k such that i is divisible by 2^k, we need to reveal the noisy version of

$$\sum_{j=1}^{2^k} x_{i-2^k+j}$$

The noise level that we have to use, is

$$\max_{j=1}^{2^k} L_{i-2^k+j}$$

because it has to be enough to hide all the x_{i-2^k+j}. The value of x_i affects at most $\log_2 n$ revealed values. The noise level L_i potentially affects (if it happens to be the maximum) the same $\log_2 n$ revealed values. Changing row i by distance d changes each revealed value affected by x_i, by DP-distance at most $\frac{\epsilon}{\log_2 n} \cdot d$, thus the whole output time series is changed by DP-distance at most $\epsilon \cdot d$. Thus we get ϵ-DP.

Using public tables. Suppose now that in addition to the single table t, the query can use some other tables $t_1, \ldots, t_m$ but these tables are all public and constant.

$$\texttt{SELECT SUM}(f(a_1, \ldots, a_S))\ \texttt{FROM}\ t, t_1, \ldots, t_m\ \texttt{WHERE}\ p(a_1, \ldots, a_S)$$

Let the columns $a_1, \ldots, a_s$ be from table t and $a_{s+1}, \ldots, a_S$ from the public tables.

Let us compute the change x_i in the query result when a row $(a_1, \ldots, a_s)$ is added to table t at time point i. The row $(a_1, \ldots, a_s)$ is joined with rows from the public tables getting (0 or more) rows of the form $(a_1, \ldots, a_s, \ldots, a_S)$ where the $a_1, \ldots, a_s$ are the same for all obtained rows but $a_{s+1}, \ldots, a_S$ may be different. We apply the filter p so that only rows satisfying p remain. Then we apply f to each row and sum the results. The obtained sum x_i is uniquely determined by $a_1, \ldots, a_s$, i.e. there exists a function g that computes this sum $g(a_1, \ldots, a_s)$. Thus the query can still be expressed in the form

$$\texttt{SELECT SUM}(g(a_1, \ldots, a_s))\ \texttt{FROM}\ t$$

using only the table t.

It may be inconvenient to express the function g in SQL. Thus we may still compute it as described above. Let us now compute the sensitivity w.r.t. the row i. We still take the 0 or more rows of the form $(a_1, \ldots, a_s, \ldots, a_S)$ where the $a_1, \ldots, a_s$ are the same for all of them. We apply the filter p, which we assume to use only public columns, private filters should be included in the function f. For each row $(a_1, \ldots, a_s, \ldots, a_S)$, we compute the sensitivity of $f(a_1, \ldots, a_s, \ldots, a_S)$ w.r.t. $(a_1, \ldots, a_s)$. Then we sum the sensitivities and obtain c_i, the sensitivity w.r.t. row i. Once we have the c_i (and x_i) for all i, we proceed as in the single table case.

Using private tables with joins by user ID. Suppose now that in addition to the single table t, the query can use some other *private* tables $t_1, \ldots, t_m$ (these tables may be used more than once, unlike table t) but these must be joined to the table t using the user ID column which should be included for all private tables and denotes the provenance of the row. Also, the user ID of each row added to table t must be different. Rows are still added only to table t, other private tables are constant (actually we can change the rows belonging to those users who are not in table t yet, as these do not affect the query result; as soon as a row from that user is added to table t, the rows of that user cannot change anymore).

Then we can join the row added at time i to other tables and obtain 0 or more rows, all of which contain private data only from the same provenance as the added row. We can compute the sensitivity of the function f applied to each obtained row w.r.t. each used row in the original tables (note that a row in the original tables is not necessarily used in all obtained rows; when it is not used the respective sensitivity is 0). Then sum the sensitivities over obtained rows to get the total sensitivity of x_i w.r.t. each used row in the original tables (if that used row is from a table used more than once then the sensitivities w.r.t. each copy of the row must be summed). Then these sensitivities are combined according to the norm of the Banach space (the Banach space is over all of the private tables, the norms of individual tables may be combined e.g. by ℓ_1-norm) to obtain c_i.

Because the user ID of each row added to table t is different and the rows joined to it have the same user ID, the budget does not have to be split between different added rows as each row uses data from a different provenance.

Adding more than one row from the same user. So far at most one row was added from each user into table t. Now consider the case where more than one row is added from the same user.

If those rows are added at the same time then we can compute their sensitivities w.r.t. each input row and combine them according to the norm of the Banach space. The noise is added to the total change in query result from adding all the rows, not to the change from each added row separately.

If the rows are not added at the same time, then the noise must be added to each change separately and we need to somehow split the budget between the changes. If we know that the user adds rows to table t at most m times then we can use $1/m$ part of the initial budget for each time, making noise m times higher.

If there is no upper bound on the number of times but we estimate it to be at most m with high probability then we can use $1/m$ part of the initial budget for each time and after the m^{th} time we can exclude any further rows added from that user.

Personalized differential privacy. So far, rows of different users could not be joined because then a row could affect the change in query result not only at the time when it is added but also at some later time points. Its privacy budget would be used at all these time points. To keep track of the budgets of all rows, we use *personalized differential privacy*. Then, at each time point or period we find the rows that are joined to some row added at that time, and update their budgets.

In addition to joining rows of different users, we can now also add rows to multiple tables and use those tables more than once. We can now also add rows to public tables, which also consumes some of the budget of the private rows added earlier to which the public rows are joined.

To allow a user to own more than one row (possibly in different tables) using a single budget, we add provenances to the personalized differential privacy, i.e. each row has a provenance and each provenance has a budget.

Removing and changing rows. So far rows could only be added. Personalized differential privacy also allows removing and changing them. Each row can now have addition time and removal time (the latter must be later than the former, otherwise the row will never be added). Query result will be changed and budget will be consumed at both of those times. If we want a row not to be removed at all then we may set the removal time very large (larger than the end of the time period for which we want differentially private updates of the query result).

To change a row, we will have a separate row for each version of the row, one will be removed and another added at the same time point, and both will have the same provenance. Because the addition and removal occur at the same time point, the budget will be consumed only once.

When revealing query result changes in longer time periods (of power-of-two length), we can ignore those rows (or row versions) that are added after the period starts but removed again before the period ends. Those rows will not affect the revealed value or its noise level and their budget is not consumed. Similarly, if a row is changed multiple times during the period, the budget will still be consumed only once.

3.3.12.5 Sensitivity w.r.t. Changes Inside Rows and Row Multiplicities. Banach analysis assumes that the join graph (which rows are joined with which) is public, only the content of the rows is private. This is the case both for single-query and for time-series analysis. For the time-series analysis, we also have times when rows are added/removed/changed, and these are also considered public. For the single-query analysis, we used *combined sensitivity* to make the join graph private.

To adapt combined sensitivity to the time-series analysis, we apply the combined sensitivity analysis separately for each time period (of power-of-two length) for which query change is revealed. This makes both the join graph and the times of changes private. But the personalized privacy budgets also depend on the join graph and change times, thus the budgets must become private as well. So we cannot reveal the amount of budget used anymore without adding noise to it and using more budget.

Leaks through the noise level. Unlike non-combined Banach sensitivity, with combined sensitivity it is possible that the noise level of a revealed value depends on a certain row but the value itself (before adding noise) does not. Budget must be used also in those cases, making the amount of budget needed much higher than for non-combined sensitivity. If the row r is in the database in the time period $[t_1, t_2]$ then at any time point $t \in [t_1, t_2]$, the sensitivity w.r.t. adding to another table a row r', depends on whether r can be joined with r', i.e. on the join key of r. Thus the budget of r is used at every time point in $[t_1, t_2]$.

The total amount of budget needed for a row r is approximately $(s + m \log_2 n)\epsilon$ where s is the total number of time points when (a version of) r is in the table, m is the number of versions of r (if the same version is removed and later added again then it is considered a new version), n is the total number of analyzed time points, ϵ is the budget used for a single query at a time point.

The noise level is $\log_2 n$ times higher than the naive method (which computes the query result separately at each time point, using $n\epsilon$ budget in total) because the query result at each time point is computed as a sum of up to $\log_2 n$ values. Thus we may actually use only $n\epsilon / \log_2 n$ budget for the naive method to get the same noise level.

Using global sensitivity. To avoid noise levels depending on input rows, we can use global sensitivity instead of local. We combine global Banach sensitivity and global adding/removing rows sensitivity. The latter requires knowing in advance how many times each row can be used in total over all time points. This may be possible for real-time analysis if there are some restrictions on the input distribution, or for batch analysis, i.e. we are releasing the time series only after all the time points have passed.

With global sensitivity, we can use the same noise level for all revealed values as for the single query of the non-time-series analysis. Then the DP-distance corresponding to a change in input is at most $\log_2 n$ times higher than for the single query. The rows of the joined table are distributed between different time points but they are the same as for the single query. The sum of the changes in the query result over the time points is equal to the result of the single query. The DP-distance depends on the sum of absolute values of query changes. Let L be the maximum possible absolute value contribution of a row (of the joined table) to the query result. Then the total sensitivity (across all time points) w.r.t. adding/removing (all uses of) a row would be $ML \log_2 n$, where M is the maximum number of times a row can be used (if this is exceeded then DP is not guaranteed).

Thus we are now using a fixed amount ϵ_1 of budget per row use, not per time point. If a row r is joined with m other rows at a certain time point t then r uses $m\epsilon_1$ budget at time point t. If the global Banach sensitivity w.r.t. a row use is K then w.r.t. all uses of a row it is $MK \log_2 n$. Combined global sensitivity is then

$$C = \max\left(\frac{ML \log_2 n}{G}, MK \log_2 n\right) = M \log_2 n \cdot \max\left(\frac{L}{G}, K\right)$$

where G is the distance corresponding to adding/removing a row. The noise level for each released value is

$$\frac{M \log_2 n \cdot \max\left(\frac{L}{G}, K\right)}{\epsilon}$$

where ϵ is the total budget. As L and K are computed in the Banach analyzer and M, G, n, ϵ are also available there, the local sensitivity analyzer is not needed for this kind of combined sensitivity analysis.

We may also compute a separate global sensitivity for each table t (considering only changes inside table t):

$$C_t = M \log_2 n \cdot \max\left(\frac{L}{G_t}, K_t\right)$$

where G_t is the distance corresponding to adding/removing a row in table t and K_t is the global Banach sensitivity w.r.t. a row use in table t. The noise level must be the same as for a single global sensitivity. The amount of budget needed for each row use in table t in a time period, is then

$$\frac{C_t}{C} \cdot \frac{\epsilon}{M \log_2 n}$$

Combining with local Banach sensitivity. It seems that we can also combine the global adding/removing rows sensitivity with local Banach sensitivity. So far, in the Banach analyzer, the local Banach sensitivity was computed for all uses of a row combined. To achieve this, for each row r in an input table (or table copy for multiply used tables), the rows in the joined table that depend on r (let the number of such rows be m), have their norms combined using the ℓ_∞ norm instead of the original norm, because making a change with distance d in row r corresponds to making the change (with the same distance d) in m rows of the joined table. To make those m changes of distance d have the total distance equal to the original distance d, we must use the ℓ_∞ norm to join them.

Now we need to compute the local Banach sensitivity per row use, i.e. for a single row in the joined table, as is already done for the global Banach sensitivity K. Here those m changes of distance d are allowed to have the total distance up to md because the allocated budget is multiplied by the number of times a row is used, i.e. m. Thus we use the ℓ_1 norm to combine the m rows in the joined table that depend on r.

For smoothing the sensitivity, we must take into account that the local Banach sensitivity can jump to the global one with an addition/removal of a single row. Thus the β-smooth Banach sensitivity κ w.r.t. a single row use can vary between $e^{-\beta G}K$ and K. The local Banach sensitivity can be computed for each time period separately. Because only M uses of a row can contribute to the query result, the amount of budget allowed for each time period is $\epsilon_0 = \frac{\epsilon}{M \log_2 n}$ where ϵ is the total budget. This amount is actually available for each use of a row in a time period, instead of all uses (similarly to the L above being for a row in the joined table, which corresponds to a single use of a row).

ϵ_0 also limits the possible values of β. Let $\epsilon = \epsilon_b + \epsilon_\beta$, where ϵ_b is the part of ϵ used to hide the query result and ϵ_β is the part of ϵ used to hide the noise level. Then

$$\beta = \frac{\epsilon_\beta}{(\gamma + 1)M \log_2 n}$$

$$b = \frac{\epsilon_b}{(\gamma + 1)M \log_2 n}$$

Thus β will usually be very small, making it difficult to get an advantage over global sensitivity, unless G is very high. Adding/removing all uses of a row can cause the noise level of up to $M \log_2 n$ released values to change. Thus the β-smooth Banach sensitivity κ w.r.t. a single row use can actually only be allowed to vary between $e^{-\beta G}K$ and K.

Thus the noise level for a released value for a time period with smooth Banach sensitivity $\kappa \in [e^{-\beta G}K, K]$, is

$$\frac{\max\left(\frac{L}{G}, \kappa\right)}{b} = \frac{(\gamma+1)M \log_2 n \cdot \max\left(\frac{L}{G}, \kappa\right)}{\epsilon_b}$$

which can be up to $(\epsilon/\epsilon_b)(\gamma+1)$ times higher than with global Banach sensitivity. If κ may be smaller than $e^{-\beta G}K$ (it cannot be larger than K) then the noise level would be

$$\frac{(\gamma+1)M \log_2 n \cdot \max\left\{\frac{L}{G}, \kappa, e^{-\beta G}K\right\}}{\epsilon_b}$$

Compared to the combined sensitivity without time series, the noise level is about $\log_2 n$ times higher (this is for released values; for actual time series elements, it is $(\log_2 n)^{1.5}$ times) but it can be even more because the row that has the maximum κ may be used less than M times and the non-time-series noise level will then be less than $M\kappa/\epsilon_1$. Even larger increase in the noise level can be caused by the term

$$e^{-\beta G}K = e^{-\frac{\epsilon_\beta G}{(\gamma+1)M \log_2 n}}K$$

which would be

$$e^{-\frac{\epsilon_\beta G}{\gamma+1}}$$

without time series. Thus G would have to be $M \log_2 n$ times higher than for non-time-series analysis to be able to get the same advantage over global sensitivity. It can also be difficult to guess a good value of M, and a non-optimal M can further reduce accuracy.

If a table is used more than once then the G above does not need to be divided by the number of times the table is used.

Adding or removing a row r can change the noise level of those time periods where r is used, from $e^{-\beta G}K$ to K, a factor of $e^{\beta G}$. This changes the distribution of each affected released value by DP-distance $(\gamma+1)\beta G$. The number of released values affected by adding or removing r, is $M \log_2 n$. Thus the total DP-distance by which the output distribution changes, is

$$(\gamma+1)\beta G \cdot M \log_2 n = \epsilon_\beta G$$

We have implemented the method for both global and local Banach sensitivity. It only works correctly if the upper bound M cannot be exceeded. If it is exceeded then DP is not guaranteed.

3.3.12.6 Budget Renewal. If we want to release more than one value and want all the released values together to be B-differentially private then our privacy budget is B. A simple budget management algorithm would start from budget B and reduce it by the amount of budget used each time a new value is released, and checking that the budget would not go below zero. If each released value does not depend on all input rows or its sensitivity w.r.t. different input rows is different then we may benefit from more sophisticated budget management techniques, two of which are discussed in this section.

We discuss how to take advantage of the independence of changes even when we release query results and not changes. This approach can be used improve the accuracy of the Kalman filter. However, as in Sec. 3.3.12.3, we lose the privacy guarantee if the changes are not actually independent. The approach takes the independence into account then tracking the privacy budget. The part of budget used for releasing the parts of the values released in the past that are independent of values released in the future, is returned to the user and can be reused in the future. This only ensures differential privacy of each value at the moment when it is released. Values released in the future may weaken the privacy. To

ensure that privacy continues into future, we must in addition to tracking the amount of *used budget*, track the amount of *remaining budget*. The part of budget that will be used for releasing the parts of the values released in the future that are independent of values released in the past, is added to the remaining budget, but not above the initial budget. Every time the budget is used, the remaining budget must not go below zero and used budget must not go above the initial budget. This ensures differential privacy over the whole time period. The privacy budget seems to renew over time and this is due to the independence in the changes, not because the data subjects agree to give up some privacy over time.

For long time series, the amount of privacy budget needed to release them differentially privately with a given accuracy (noise level) grows by the number of time points n, either polylogarithmically (as in Sec. 3.3.12.3) or faster ($\Theta(\sqrt{n})$ to $\Theta(n)$ for the Kalman filter). In this section, we try to find some motivation for why the budget could renew.

Two-element time series. Let

$$X, Y \sim N(0,1) \text{ and independent}$$

$$Z = \frac{X+Y}{\sqrt{2}}$$

Then

$$Z \sim N(0,1)$$

The random values X and Z can be viewed as two successive elements of a time series. The distribution of Z depends only on the value of X. Thus the time series is a Markov chain. Let

$$W = X\sqrt{2} - Z$$

Then

$$X = \frac{Z+W}{\sqrt{2}}$$

$$W = \frac{X-Y}{\sqrt{2}}$$

$$W \sim N(0,1)$$

Lemma 3.102. *W is independent of Z.*

Proof.

$$f_X(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

$$f_Y(y) = \frac{1}{\sqrt{2\pi}} e^{-y^2/2}$$

$$f_{X+Y}(u) = \int_{-\infty}^{\infty} f_X(x) f_Y(u-x)dx = \int_{-\infty}^{\infty} \frac{1}{2\pi} e^{-x^2/2} e^{-(u-x)^2/2} dx$$

$$= \int_{-\infty}^{\infty} \frac{1}{2\pi} e^{-(x^2+u^2-2ux+x^2)/2} dx$$

$$= e^{-u^2/4} \int_{-\infty}^{\infty} \frac{1}{2\pi} e^{-(u^2/2-2ux+2x^2)/2} dx$$

$$= e^{-u^2/4} \int_{-\infty}^{\infty} \frac{1}{2\pi} e^{-(u^2/2-2ux+2x^2)/2} dx$$

$$= e^{-u^2/4} \int_{-\infty}^{\infty} \frac{1}{2\pi} e^{-(u^2/4-ux+x^2)} dx$$

$$= e^{-u^2/4} \int_{-\infty}^{\infty} \frac{1}{2\pi} e^{-(x-\frac{u}{2})^2} dx$$

$$= e^{-u^2/4} \int_{-\infty}^{\infty} \frac{1}{2\pi} e^{-(\frac{x-\frac{u}{2}}{1/\sqrt{2}})^2/2} dx$$

$$= \frac{1}{\sqrt{2\pi}} e^{-u^2/4} \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-(\frac{x-\frac{u}{2}}{1/\sqrt{2}})^2/2} dx$$

$$= \frac{1}{\sqrt{2} \cdot \sqrt{2\pi}} e^{-u^2/4} \int_{-\infty}^{\infty} \frac{1}{(1/\sqrt{2})\sqrt{2\pi}} e^{-(\frac{x-\frac{u}{2}}{1/\sqrt{2}})^2/2} dx$$

$$= \frac{1}{\sqrt{2} \cdot \sqrt{2\pi}} e^{-u^2/4} dx$$

$$= \frac{1}{\sqrt{2} \cdot \sqrt{2\pi}} e^{-(u/\sqrt{2})^2/2} dx$$

$$f_{X+Y,X-Y}(u,v) du\, dv = f_X(\frac{u+v}{2}) f_Y(\frac{u-v}{2}) dx\, dy$$

$$du\, dv = \frac{1}{2} dx\, dy$$

$$f_{X+Y,X-Y}(u,v) = \frac{1}{2} f_X(\frac{u+v}{2}) f_Y(\frac{u-v}{2})$$

$$= \frac{1}{2} \cdot \frac{1}{\sqrt{2\pi}} e^{-(\frac{u+v}{2})^2/2} \cdot \frac{1}{\sqrt{2\pi}} e^{-(\frac{u-v}{2})^2/2}$$

$$= \frac{1}{4\pi} e^{-(\frac{u+v}{2})^2/2 - (\frac{u-v}{2})^2/2}$$

$$= \frac{1}{4\pi} e^{-(u+v)^2/8 - (u-v)^2/8}$$

$$= \frac{1}{4\pi} e^{-(2u^2+2v^2)/8}$$

$$= \frac{1}{4\pi} e^{-(u^2+v^2)/4}$$

$$= \frac{1}{\sqrt{2} \cdot \sqrt{2\pi}} e^{-(u/\sqrt{2})^2/2} \cdot \frac{1}{\sqrt{2} \cdot \sqrt{2\pi}} e^{-(v/\sqrt{2})^2/2}$$

$$= f_{X+Y}(u) f_{X-Y}(v)$$

This proves that $X + Y$ and $X - Y$ are independent. Thus also Z and W are independent. □

Thus the reverse of the time series is also a Markov chain with the same distribution.

n-element time series. We can now describe the whole time series $X_1, \ldots, X_n$:

$$X_1 \sim N(0, 1)$$

$$Y_i \sim N(0, 1) \text{ for all } i = 1, \ldots, n$$

X_1 and all Y_i are mutually independent.

$$X_{i+1} = \frac{X_i + Y_i}{\sqrt{2}}$$

Then we have

$$X_i \sim N(0, 1)$$

From the results above we also have

$$X_n \sim N(0, 1)$$

$$W_i \sim N(0, 1) \text{ for all } i = 1, \ldots, n$$

X_n and all W_i are mutually independent.

$$X_{i-1} = \frac{X_i + W_i}{\sqrt{2}}$$

Differential privacy. Suppose we add Laplace noise with level L_i to each X_i and reveal the noisy result:

$$N_i \sim Lap(L_i)$$

$$R_i = X_i + N_i$$

Then R_i is $\frac{1}{L_i}$-differentially private w.r.t. X_i. Let $\epsilon_i = \frac{1}{L_i}$. Then we may say that the amount of budget used at time point i is ϵ_i.

However, the X_i are not independent and thus the other R_j may also leak information about X_i. Suppose that $j > i$. Then

$$R_j = X_j + N_j = X_{j-1}(\sqrt{2})^{-1} + Y_{j-1}(\sqrt{2})^{-1} + N_j$$

$$= X_{j-2}(\sqrt{2})^{-2} + Y_{j-2}(\sqrt{2})^{-2} + Y_{j-1}(\sqrt{2})^{-1} + N_j = \ldots$$

$$= X_{j-k}(\sqrt{2})^{-k} + Y_{j-k}(\sqrt{2})^{-k} + \ldots + Y_{j-m}(\sqrt{2})^{-m} + \ldots + Y_{j-1}(\sqrt{2})^{-1} + N_j = \ldots$$

$$= X_i(\sqrt{2})^{i-j} + N_j + \sum_{m=1}^{j-i} Y_{j-m}(\sqrt{2})^{-m}$$

Note that $X_i(\sqrt{2})^{i-j}$ is $(\sqrt{2})^{i-j}$-sensitive w.r.t. X_i, thus $X_i(\sqrt{2})^{i-j} + N_j$ is $\frac{(\sqrt{2})^{i-j}}{L_j}$-differentially private w.r.t. X_i. Because $\sum_{m=1}^{j-i} Y_{j-m}(\sqrt{2})^{-m}$ is independent of X_i and N_j, adding it does not weaken differential privacy, thus R_j is also $\frac{(\sqrt{2})^{i-j}}{L_j}$-differentially private w.r.t. X_i.

If $j < i$ then we get

$$R_j = X_i(\sqrt{2})^{j-i} + N_j + \sum_{m=1}^{i-j} W_{j+m}(\sqrt{2})^{-m}$$

and R_j is $\frac{(\sqrt{2})^{j-i}}{L_j}$-differentially private w.r.t. X_i.

Thus for all j, R_j is $\frac{(\sqrt{2})^{-|i-j|}}{L_j}$-differentially private w.r.t. X_i. We may also express it as $(\sqrt{2})^{-|i-j|}\epsilon_j$-differentially private w.r.t. X_i. The whole noisy time series $(R_1, \ldots, R_n)$ is $\sum_{j=1}^{n}(\sqrt{2})^{-|i-j|}\epsilon_j$-differentially private w.r.t. X_i.

Thus the total amount of budget used is not $\sum_{j=1}^{n} \epsilon_j$ but only

$$B = \sum_{j=1}^{n} (\sqrt{2})^{-|i-j|} \epsilon_j$$

We may say that the budget *renews* to some extent. If $j < i$ then the budget used at time point j contributes $\sqrt{2}$ times less to the total used budget than the same amount of budget used at time point $j+1$. The total amount of budget used by time j can be defined as

$$E_j = \sum_{k=1}^{j} (\sqrt{2})^{k-j} \epsilon_k$$

Then $E_{j+1} = \frac{E_j}{\sqrt{2}} + \epsilon_{j+1}$. Thus the amount of budget used is divided by $\sqrt{2}$ between time points j and $j+1$, which can be viewed as budget renewal. This renewal works until time point i, when E_i becomes equal to the sum of the first i terms of B.

Let us now consider what happens after time point i. If $j > i$ then the budget used at time point j contributes $\sqrt{2}$ times less to the total used budget than the same amount of budget used at time point $j-1$. The total amount of budget used from time j on can be defined as

$$F_j = \sum_{k=j}^{n} (\sqrt{2})^{j-k} \epsilon_k$$

This amount of budget must be available at time point j before using the ϵ_j at that time point. Then $F_j = (F_{j-1} - \epsilon_{j-1})\sqrt{2}$. Thus the amount of budget available for future use is multiplied by $\sqrt{2}$ between time points $j-1$ and j. This is also a kind of budget renewal but slightly different from the one that occurs before time point i. At time point i, F_i is equal to the sum of the last $n-i+1$ terms of B.

Altogether,

$$B = E_i + F_i - \epsilon_i$$

The ϵ_i is subtracted because it is included in both E_i and F_j but only once in B. Thus we can use the described kinds of budget renewal using the following budget tracking algorithm:

- Let $E_0 = 0$.
- For each time point $j = 1, \ldots, i-1$:
  - Let ϵ_j be the amount of budget used at time point j.
  - Let $E_j = \frac{E_{j-1}}{\sqrt{2}} + \epsilon_j$.
  - If $E_j > (\sqrt{2})^{i-j} B$ then **fail**.
  - Reveal R_j.
- For time point $j = i$:
  - Let ϵ_j be the amount of budget used at time point j.
  - Let $E_j = \frac{E_{j-1}}{\sqrt{2}} + \epsilon_j$.
  - If $E_j > B$ then **fail**.
  - Let $F_{j+1} = (B - E_j)\sqrt{2}$.
  - Reveal R_j.
- For each time point $j = i+1, \ldots, n$:
  - Let ϵ_j be the amount of budget used at time point j.
  - Let $F_{j+1} = (F_j - \epsilon_j)\sqrt{2}$.
  - If $F_{j+1} < 0$ then **fail**.
  - Reveal R_j.

If this algorithm does not **fail** then the revealed values together are B-differentially private w.r.t. X_i. If it does **fail** then the revealed values before the **fail** together are B-differentially private w.r.t. X_i.

Differential privacy w.r.t. all X_i. The above algorithm only guarantees differential privacy w.r.t. X_i for a single i. Suppose we want that for all i, the revealed values together are B-differentially private w.r.t. X_i. One way to achieve this is to run n instances of the above algorithm in parallel, one instance (thread i) for each i to achieve differential privacy w.r.t. X_i. The threads (instances) are synchronized at each "Reveal R_j" statement, which is executed only when none of the threads has **fail**ed by that point.

Let us try to combine this into a sequential algorithm. Note that for all j, ϵ_j is the same in all threads. Thus also E_j is the same in all threads where it is defined. The check $E_j > (\sqrt{2})^{i-j}B$ is performed in threads $i \geq j$. To check if at least one of the threads fails here, we may check $E_j > B$.

Now consider F_j, which is not necessarily the same in all threads. It is, however, updated by the monotonic rule

$$F_{j+1} = (F_j - \epsilon_j)\sqrt{2}$$

thus if we have the F_j that is the minimum F_j among the threads where it is defined then F_{j+1} will also be the minimum among those threads. At each time point j there is one thread (thread j) where an F value is defined for the first time. To take this into account, the minimum F_{j+1} is computed as follows:

$$F_{j+1} = \min(F_j - \epsilon_j, B - E_j) \cdot \sqrt{2}$$

Once we have the minimum F_{j+1}, we can check $F_{j+1} < 0$ to test whether at least one of the threads fails at that check. Note that this check is only executed when $E_j \leq B$ thus it can be replaced with

$$F_j - \epsilon_j < 0$$

We get the following algorithm:

- Let $E_0 = 0, F_1 = B$.
- For each time point $j = 1, \ldots, n$:
  - Let ϵ_j be the amount of budget used at time point j.
  - Let $E_j = \frac{E_{j-1}}{\sqrt{2}} + \epsilon_j$.
  - If $E_j > B$ then **fail**.
  - If $F_j - \epsilon_j < 0$ then **fail**.
  - Reveal R_j.
  - Let $F_{j+1} = \min(F_j - \epsilon_j, B - E_j) \cdot \sqrt{2}$.

Let us write it in the imperative style, where the values of variables can change:

- Let $E := 0, F := B$.
- For each time point $j = 1, \ldots, n$:
  - Let $E := \frac{E}{\sqrt{2}}$.
  - Let ϵ be the amount of budget used at time point j.
  - Let $E := E + \epsilon$.
  - Let $F := F - \epsilon$.
  - If $E > B$ then **fail**.
  - If $F < 0$ then **fail**.
  - Reveal R_j.
  - Let $F := \min(F, B - E)$.
  - Let $F := F\sqrt{2}$.

This shows most clearly how the budget renewal works. Between any two successive time points, the used budget E is divided by $\sqrt{2}$ and the available budget F is multiplied by $\sqrt{2}$.

If, before the renewal, $E + F > B$, the F is reduced until $E + F = B$. This ensures that the total budget (already used + available for future use) is never larger than B at any time.

The used budget by itself must not become larger than B and the available budget must not become negative. This sets the restrictions on how much budget can be used at any given time point. Note that this does not show the optimal use of the budget to get the best accuracy. It only ensures differential privacy.

Optimizing the accuracy. Let us try to get the best accuracy within the given limits. Before each renewal, we must have $E + F \leq B$, so let us assume that $E + F = B$ to have the largest renewal potential. The $E+F$ renews to $\frac{E}{\sqrt{2}}+F\sqrt{2}$. If we want to get back to the previous E and F, we can use $\epsilon = (1-\frac{1}{\sqrt{2}})E$ and $(\sqrt{2}-1)F \geq \epsilon$. We get

$$3\epsilon \leq (\sqrt{2}-1)(E+F) = (\sqrt{2}-1)B$$

$$\epsilon \leq \frac{\sqrt{2}-1}{3}B$$

Thus we can use up to $\frac{\sqrt{2}-1}{3}B$ or about 13.8% of the whole budget at each time point of an infinite time series. For a finite time series, we can use a bit more if we use up the budget over the finite time, so that in the end we have $E = B$ and $F = 0$.

Generalizations. We have investigated the case where the time series we want to reveal is a Markov chain (also in the opposite direction) whose elements are all from the standard normal distribution and the underlying value changes over time with a specific speed. It seems possible to generalize our results to the case where the elements of the time series are from other normal distributions and different elements may be from different normal distributions, and where the underlying value may change at different speeds.

Distance for differential privacy. Above we showed how to get differential privacy w.r.t. each X_i. We however assumed a certain distribution of $(X_1, \ldots, X_n)$ but differential privacy should be independent of the input distribution. Thus the input there was not actually $(X_1, \ldots, X_n)$ but $(W_2, \ldots, W_i, X_i, Y_i, \ldots, Y_{n-1})$ when considering differential privacy w.r.t. X_i.

Thus the input is different for each X_i but each of these inputs (and also $(X_1, \ldots, X_n)$) uniquely determines all the others. We would like to use only $(X_1, \ldots, X_n)$ as the input an achieve the same results. Thus we need to define a distance on the $(X_1, \ldots, X_n)$. Let h_i be the function that converts $(X_1, \ldots, X_n)$ to the corresponding $(W_2, \ldots, W_i, X_i, Y_i, \ldots, Y_{n-1})$:

$$h_i(X_1, \ldots, X_n) = (W_2, \ldots, W_i, X_i, Y_i, \ldots, Y_{n-1})$$

where

$$Y_j = X_{j+1}\sqrt{2} - X_j$$

$$W_j = X_{j-1}\sqrt{2} - X_j$$

Let H_i be the inverse of this function:

$$H_i(W_2, \ldots, W_i, X_i, Y_i, \ldots, Y_{n-1}) = (X_1, \ldots, X_n)$$

where

$$X_{j+1} = \frac{X_j + Y_j}{\sqrt{2}} \text{ for } j = i, i+1, \ldots, n-1$$

$$X_{j-1} = \frac{X_j + W_j}{\sqrt{2}} \text{ for } j = i, i-1, \ldots, 2$$

Another way to express X_j here is

$$X_j = X_i(\sqrt{2})^{i-j} + \sum_{m=1}^{j-i} Y_{j-m}(\sqrt{2})^{-m} \text{ if } j > i$$

$$X_j = X_i(\sqrt{2})^{j-i} + \sum_{m=1}^{i-j} W_{j+m}(\sqrt{2})^{-m} \text{ if } j < i$$

Then the distance d on the $(X_1, \ldots, X_n)$ must satisfy

$$d(H_i(W_2, \ldots, W_i, X_i, Y_i, \ldots, Y_{n-1}), H_i(W_2, \ldots, W_i, X_i + c, Y_i, \ldots, Y_{n-1})) = |c|$$

since here we modify X_i by $|c|$ and recompute the X_i where $i \neq j$.

Let $g_{i,c}$ be the following function:

$$g_{i,c}(W_2, \ldots, W_i, X_i, Y_i, \ldots, Y_{n-1}) = (W_2, \ldots, W_i, X_i + c, Y_i, \ldots, Y_{n-1})$$

Note that

$$(H \circ g_{i,c} \circ h)(X_1, \ldots, X_n) = (X'_1, \ldots, X'_n)$$

where

$$X'_j = X_j + \frac{c}{(\sqrt{2})^{|i-j|}}$$

This describes a change in $(X_1, \ldots, X_n)$ corresponding to distance $|c|$. We may actually allow

$$|X_j - X'_j| \leq \frac{c}{(\sqrt{2})^{|i-j|}}$$

without increasing the distance $|c|$ because reducing the change in X_j or reversing the direction of the change does not increase the DP-distance change in the distribution of the released output R_j.

We have defined the distance d for only some pairs of values. To make it complete, we take the transitive closure. Then using the budget tracking algorithm described earlier, we can guarantee B-differential privacy w.r.t. distance d (if the algorithm does not fail).

3.3.12.7 Enforcing Budget Limits. In this section, we will see in which cases the budget limits can be enforced by excluding the rows whose budget becomes exhausted (as opposed to some known restrictions on the input distribution guaranteeing an upper bound on the amount of budget used). We will also consider the case where budgets can be increased over time.

Banach sensitivity with enforced budget limits. When using only Banach sensitivity (sensitivity w.r.t. changes inside rows), the budgets are public because changes inside rows do not change the amount of budget used. (The amount of budget used depends on the join keys but changing the join key is considered as exchanging the whole row instead of as a change inside the row.) Thus we can exclude rows whose budget is exhausted and still retain DP. The excluded rows also do not need to be taken into account when computing the sensitivity.

Adding/removing rows sensitivity with enforced budget limits. So far we could only compute (without revealing) the amount of budget used but could not guarantee that a given limit is not exceeded.

Let us use the following mechanism. Consider the query that counts the number of rows in the table (let this be the *joined table*) obtained by joining the N tables together in the way described above. To compute this query differentially privately, we use a personalized budget ϵ_i for each r_i. Let ϵ_0 be the amount of budget needed for each use of r_i. Thus r_i can be used $\lfloor \epsilon_i/\epsilon_0 \rfloor$ times.

Consider a row R in the joined table. Let the rows from which it is composed be $r_{s_1}, \ldots, r_{s_N}$ where $\tau_{s_i} = i$. Sort the list $(s_1, \ldots, s_N)$ in decreasing order to get $(s'_1, \ldots, s'_N)$, which we call the *timestamp* of R. We use the lexicographic order on timestamps to order rows of the joined table by time. The first component of the timestamp is the time point when a row of the joined table is used, the rest of the components are used to order rows used at the same time point.

We sort the rows of the joined table in increasing order of timestamps. Then we go through the rows is this order. For each row R, we check if all the component rows of R have at least ϵ_0 budget left. If yes, we include R in the query result and decrease the budget of each component row by ϵ_0. If no, we do not include R in the query result and do not change budgets.

However, this mechanism cannot work for the most general class of queries because we have the following counterexample:

```
SELECT * FROM t1, t2 WHERE t2.id = t1.id OR t2.id = t1.id + 1
```

Each of the `id` columns is a primary key of its table but due to the `OR` in the condition, each row of either table can be joined to up to 2 rows of the other table. Suppose that each table contains rows with `id` $= 1, 2, \ldots, k$ and each row has budget for only one use. Then the joined rows that are included, are the ones with `t2.id = t1.id`. If we now remove from the second table the row with `id` $= 1$ then the joined rows that are included, will be the ones with `t2.id = t1.id + 1`, i.e. the ones previously excluded. Thus $k-1$ rows become included and k rows become excluded, for a total sensitivity of $2k-1$. The necessary DP noise makes the differentially private result useless.

Let us try if the mechanism can achieve DP for some special cases where the query is restricted in some way.

$1:n$ **Joins.** Let there be N tables numbered $1, \ldots, N$. Let $r_1, \ldots$ be rows that are added to the tables in that order, i.e. r_i is added at time point i. Let τ_i be the number of the table into which r_i is added. The N tables are joined together so that for each i, tables i and $i+1$ are joined with $1:n$ join, each row of table $i+1$ can be joined with only one row of table i but a row of table i can be joined with any number of rows of table $i+1$. Let $r_{\rho(i)}$ be the unique row of table $\tau_i - 1$ to which r_i can be joined. If $\tau_i = 1$ then let $\rho(i) = 0$.

Now let us consider what is the sensitivity of the query result computed with the mechanism proposed in the previous paragraph w.r.t. removing a row r_k. Removing r_k removes all uses of r_k in the joined table that where included in the query result. Consider one such use, i.e. a row R in the joined table.

Lemma 3.103. *Removing R cannot cause more than one other row to become included (unless budgets can be increased over time).*

Proof. Let the rows from which R is composed be $r_{s_1}, \ldots, r_{s_N}$ where $\tau_{s_i} = i$. If we remove R then each r_{s_i} will have budget for one extra use. Suppose that this causes a row R' to be included in the query. Let the rows from which it is composed be $r_{s'_1}, \ldots, r_{s'_N}$ where $\tau_{s'_i} = i$. Because R' was included only after R was removed, there must be some $r_{s'_m}$ which obtained the budget that was previously used by R, i.e. $s'_m = s_m$. We also have $s'_{m-1} = \rho(s'_m) = \rho(s_m) = s_{m-1}$. Similarly we get $s'_j = s_j$ for all $j = m, m-1, \ldots, 1$. Also R' must have a later timestamp than R, otherwise R' would have been included previously instead of R.

Suppose that in addition to R', there is another row R'' included when R is removed. W.l.o.g. let the timestamp of R'' be later than that of R' (they both must be later than R). Let the rows from which R'' is composed be $r_{s''_1}, \ldots, r_{s''_N}$ where $\tau_{s''_i} = i$. Similarly to above we get that for some $\ell \geq 1$, we get $s''_j = s_j$ for all $j = \ell, \ell-1, \ldots, 1$.

Consider the case $\ell \leq m$. Then $s''_\ell = s_\ell = s'_\ell$. Excluding R allowed R'' to be included because it obtained the budget for $r_{s'_\ell}$ from R. But R' also consumes the budget for $r_{s''_\ell} = r_{s'_\ell}$ and because R' has an earlier timestamp than R'', this budget released from R never reaches R''. Thus removing R cannot cause R'' to become included.

Now consider the case $\ell > m$. Then $s''_m = s_m = s'_m$. Excluding R allowed R'' to be included because it obtained the budget for $r_{s''_\ell}$ from R. This budget is not consumed by R'. But R' consumes the budget for $r_{s'_m} = r_{s''_m}$ and because R' has an earlier timestamp than R'', this budget released from R never reaches R''. Thus removing R cannot cause R'' to become included, unless some extra budget is added for $r_{s''_m}$ between the timestamps of R' and R'' and it is not allowed to use this budget to include rows of the joined table that were previously excluded.

Thus removing R can cause at most one row to become included. □

If the extra budget cannot be used for previous rows then up to T rows can become included. Indeed, the budget released from each of the T component rows of R can be used by at most one of the rows that become included, as each row use needs the same amount, ϵ_0.

Lemma 3.104. *Removing R cannot cause any other rows to become excluded (unless budgets can be increased over time).*

Proof. Suppose that there is a row R'' that becomes excluded when R is removed and R' is included. The timestamp of R'' must be later than R' because excluding R cannot use up the budgets needed by R'', only including R' can. Let the rows from which R'' is composed be $r_{s''_1}, \ldots, r_{s''_N}$ where $\tau_{s''_i} = i$. Then R' must have used up the budget of $r_{s''_\ell}$ for some ℓ. Thus $s'_\ell = s''_\ell$ and similarly to above we get that $s''_j = s'_j$ for all $j = \ell, \ell - 1, \ldots, 1$.

Consider the case $\ell \leq m$. Then $s_\ell = s'_\ell = s''_\ell$. Thus the budget of r_{s_ℓ} was previously used by R and after removing R by R', thus removing R cannot affect the budget of r_{s_ℓ} available for R'', thus it cannot cause R'' to become excluded.

Now consider the case $\ell > m$. Then $s_m = s'_m = s''_m$. The budget of r_{s_m} was previously used by R and after removing R by R', thus removing R cannot affect the budget of r_{s_m} available for R''. If there is not enough budget of r_{s_m} to include R'' after removing R, then there was not enough budget also before removing R, thus removing R cannot cause R'' to become excluded. □

Thus we can achieve DP for queries that use only $1 : n$ joins.

Increasing budgets. So far, budgets were not increased over time. The whole budget was available from the beginning. Note that budgets cannot be decreased over time because they are private and we do not know if there is enough budget left to take away. Let us now consider increasing budgets over time, i.e. between any two consecutive time points, we can add a non-negative number to each budget (the number may be different for different budgets). Also, the initial budgets (before the first time point) are non-negative.

So far we have only considered how many rows can become included or excluded but we have not counted the rows that change their inclusion time. If budgets are not changed or extra budgets cannot be used for rows excluded earlier then rows can be included only at the time they become available for inclusion (i.e. the last of its components is added to the database). Thus inclusion times cannot change in this case.

However, if extra budget can be used for previously excluded rows then removing R can change inclusion times of other rows. For example, suppose $R_1, \ldots, R_k$ are in increasing order of timestamps and all contain the row r. Initially r has budget for only one use and R uses it up. For each $i = 2, \ldots, k$, we increase the budget of r by one use between right before the timestamp of R_i (but strictly after that of R_{i-1}). Thus for each $i = 1, \ldots, k - 1$, the row R_i will be included at the timestamp of R_{i+1} and the row R_k will be excluded. Now, if we remove R, the released budget can be used to include R_k. However, this is not the only change, the rows $R_1, \ldots, R_{k-1}$ were previously included and remain included but their inclusion times change—now each R_i will be included at its own timestamp instead of that of R_{i+1}.

Thus we cannot allow using the extra budget for previously excluded rows in the joined table, i.e. for previously excluded uses of rows in the original tables. Note that the extra budget can still be used for new uses of the rows in the original tables whose previous uses were excluded. Specifically, the extra budget can be used for those and only those rows of the joined table for which at least one of its components (rows of the original tables) was added later than the extra budget.

Let us find the sensitivity of the query result w.r.t. removing a row use R when budgets can be increased. Lemma 3.103 becomes weaker:

Lemma 3.105. *If budgets can be increased then removing R cannot cause more than T other rows to become included.*

Proof. The budget ϵ_0 released from each of the T component rows of R can be used by at most one of the rows that become included, as the latter uses up the same amount, ϵ_0. □

Lemma 3.104 still holds:

Lemma 3.106. *If budgets can be increased then removing R cannot cause any other rows to become excluded.*

Proof. We use the proof of Lemma 3.104 but there are up to T possible choices for R' in the proof. At least one of them must have used up the budget of $r_{s''_\ell}$ for some ℓ. Choose one of those as R'. □

Thus the sensitivity increases from 2 to $T + 1$ when budgets can be increased.

Combining with global Banach sensitivity. So far we got DP w.r.t. adding/removing rows for $1 : n$ joins for COUNT queries. Now let us combine it with global Banach sensitivity and use SUM queries. This is similar to using global sensitivity in Sec. 3.3.12.5.

First consider the case when budgets are not increased. Let L be the maximum possible absolute value contribution of a row (of the joined table) to the query result. Then the total sensitivity (across all time points) w.r.t. adding/removing (all uses of) a row would be $2LM \log_2 n$, where M is the maximum number of times a row can be used (if this is exceeded then some row uses are excluded). The factor 2 is sensitivity of the respective COUNT query w.r.t. adding/removing rows. If the global Banach sensitivity w.r.t. a row use is K then w.r.t. all uses of a row it is $MK \log_2 n$. Combined global sensitivity is then

$$C = \max\left(\frac{2LM \log_2 n}{G}, MK \log_2 n\right) = M \log_2 n \cdot \max\left(\frac{2L}{G}, K\right)$$

where G is the distance corresponding to adding/removing a row. The noise level for each released value is

$$\frac{M \log_2 n \cdot \max\left(\frac{2L}{G}, K\right)}{\epsilon}$$

where ϵ is the total budget.

Now consider the case where budgets are increased. Then the sensitivity of the respective COUNT query will be $T + 1$. Thus the factor 2 in $2L$ will become $T + 1$. The noise level for each released value will be

$$\frac{M \log_2 n \cdot \max\left(\frac{(T+1)L}{G}, K\right)}{\epsilon}$$

Combining with local Banach sensitivity. We can also combine with local Banach sensitivity. This is similar to combining global and Banach sensitivity in Sec. 3.3.12.5. Let $\epsilon = \epsilon_b + \epsilon_\beta$, where ϵ_b is the part of ϵ used to hide the query result and ϵ_β is the part of ϵ used to hide the noise level. Then

$$\beta = \frac{\epsilon_\beta}{(\gamma + 1)M \log_2 n}$$

$$b = \frac{\epsilon_b}{(\gamma + 1)M \log_2 n}$$

Let κ be β-smooth Banach sensitivity w.r.t. a single row use. The noise level for each released value will be

$$\frac{(\gamma + 1)M \log_2 n \cdot \max\left\{\frac{2L}{G}, \kappa, e^{-\beta G} K\right\}}{\epsilon_b}$$

if budgets are constant and

$$\frac{(\gamma + 1)M \log_2 n \cdot \max\left\{\frac{(T+1)L}{G}, \kappa, e^{-\beta G} K\right\}}{\epsilon_b}$$

if budgets can be increased.

3.3.13 Model-Checking Sensitivity of SQL Queries. Consider again SQL workflows, consisting of tables and queries, as defined in Sec. 3.2.1.1. We have proposed a model-checking based approach to determine the sensitivity of such workflows, which, differently from previous approaches can provide precise bounds in certain cases.

3.3.13.1 Non-GROUP BY Queries. The SQL workflows are made up of queries. We consider two types of queries here — those with out GROUP BY, and those containing it. The first kind of the queries is defined as follows.

A *database query* $Q = (J, F, P)$ (*join—filter—project*) over the database schema *dbs* consists of the components named below. Let us use the notation defined in Sec. 3.2.1.1.

- $J \in \{t_1, \dots, t_m\}^*$, where Z^* denotes the set of all sequences of elements of the set Z. Let $|J|$ be the length of J and $J[i]$ the i-th component of J. Let $D[J]$ denote the Cartesian product $D[J[1]] \times \cdots \times D[J[|J|]]$. For $b \in D[J]$, let $b[i]$ denote the sequence of the elements of b corresponding to the table $J[i]$.
- F is a predicate on the set $D[J]$.
- P is a sequence of functions $p_1, \dots, p_r$, where $p_i : D[J] \to X_i$. The set X_i may be any set (it is the set of possible values in the ith column of the result table).

The application of a query $Q = (J, F, P)$ to a database $Y \in \mathcal{Y}$ proceeds as follows:

1. Let $A = \prod_{i=1}^{|J|} Y.J[i]$. Note that A is a multiset over $D[J]$.
2. Let $B \subseteq A$ be the multiset of all elements of A that satisfy F.
3. Output the multiset $\{\!|(p_1(b), \dots, p_r(b)) \mid b \in B|\!\}$.

Such kind of queries cover the SQL-queries containing JOIN-s and WHERE-clauses, where certain attributes of the joined tables are selected into the result. The set of projections P cover the computations over the attributes. Assuming that the predicate F and the functions $p_1, \dots, p_r$ can be specified in SQL, the query Q corresponds to the following SQL query:

$$\texttt{SELECT } \overline{[\![p_1]\!]}, \dots, \overline{[\![p_r]\!]} \texttt{ FROM } J[1], \dots, J[|J|] \texttt{ WHERE } \overline{[\![F]\!]}$$

where $\overline{[\![z]\!]}$ denotes the syntactic object (i.e. SQL expression) with semantics z.

The queries described here have the following distributivity properties, which are easy to check if one considers how a particular row may have ended up in the result of the query.

Lemma 3.107. *Let $dbs = (t_1 : r_1, \dots, t_m : r_m)$ be a database schema, $Q = (J, F, P)$ a query over dbs, and Y a database over dbs. Let $i \in \{1, \dots, m\}$ and R_1 and R_2 be relations over r_i. Let R'_1 be the result of Q on $Y[t_i \mapsto R_1]$ and R'_2 be the result of Q on $Y[t_i \mapsto R_2]$. Let R' be the result of Q on $Y[t_i \mapsto R_1 \cup R_2]$.*

(a) If J contains t_i only once, then $R' = R'_1 \cup R'_2$ and $R' \setminus R'_1 = R'_2$.

(b) Suppose that $R_1 \cap R_2 = \emptyset$. Let $i_1, \dots, i_k$ be the positions of t_i in J. Let $F_{i_j}^{R_2}$ be a predicate on $D[J]$, returning true on a row $\mathbf{r}$ iff $\mathbf{r}[i_j] \in R_2$. Let Q' be the query $(J, F \wedge \bigvee_{j=1}^{k} F_{i_j}^{R_2}, P)$. Then the result of Q' on $Y[t_i \mapsto R_1 \cup R_2]$ is $R' \setminus R'_1$.

Proof. Part (a). Let $t_i = J[j]$. Then by the description of application of queries above (and using the fact that for all $k \neq j$, $J[k] \neq t_i$), we have
$R'_1 = \{\!|(p_1(b), \dots, p_r(b)) \mid b \in \prod_{k=1}^{j-1} Y.J[k] \times R_1 \times \prod_{k=j+1}^{|J|} Y.J[k], F(b)|\!\}$,
$R'_2 = \{\!|(p_1(b), \dots, p_r(b)) \mid b \in \prod_{k=1}^{j-1} Y.J[k] \times R_2 \times \prod_{k=j+1}^{|J|} Y.J[k], F(b)|\!\}$,
$R' = \{\!|(p_1(b), \dots, p_r(b)) \mid b \in \prod_{k=1}^{j-1} Y.J[k] \times (R_1 \cup R_2) \times \prod_{k=j+1}^{|J|} Y.J[k], F(b)|\!\}$. Thus $R' = R'_1 \cup R'_2$ and (because we are using multisets, not sets) $R' \setminus R'_1 = R'_2$.
Part (b). W.l.o.g. (as reordering elements of a tuple is a set isomorphism) we can assume that $i_1 = 1, \dots, i_k = k$. Then the result of Q' on $Y[t_i \mapsto R_1 \cup R_2]$ is $R'' = \{\!|(p_1(b), \dots, p_r(b)) \mid b \in (R_1 \cup R_2)^k \times \prod_{j=k+1}^{|J|} Y.J[j], F(b) \wedge \bigvee_{j=1}^{k} (b[j] \in R_2)|\!\}$. Because for $j = 1, \dots, k$, we have $b[j] \in R_1 \cup R_2$, and $R_1 \cap R_2 = \emptyset$,

the condition $\bigvee_{j=1}^{k}(b[j] \in R_2) = \neg \bigwedge_{j=1}^{k}(b[j] \notin R_2) = \neg \bigwedge_{j=1}^{k}(b[j] \in R_1) = ((b[1], \ldots, b[k]) \notin R_1^k)$. Thus $R'' = \{\!\{(p_1(b), \ldots, p_r(b)) \mid b \in ((R_1 \cup R_2)^k \setminus R_1^k) \times \prod_{j=k+1}^{|J|} Y.J[j], F(b)\}\!\} =$
$\{\!\{(p_1(b), \ldots, p_r(b)) \mid b \in (R_1 \cup R_2)^k \times \prod_{j=k+1}^{|J|} Y.J[j], F(b)\}\!\} \setminus \{\!\{(p_1(b), \ldots, p_r(b)) \mid b \in R_1^k \times \prod_{j=k+1}^{|J|} Y.J[j], F(b)\}\!\} = R' \setminus R'_1$. □

Set semantics. When evaluating the queries in set semantics, a relation R over the schema r is a subset of $D[r]$ instead of a multiset over $D[r]$, and the multiset constructor in Step 3 of Sec. 3.3.13.1 is replaced with set constructor, i.e. duplicates are removed. We actually only require the output relation to be a set, the input relations can still be multisets.

3.3.13.2 GROUP BY Queries. A GROUP BY query $Q = (J, F, G, P_G, P_A)$ (*join—filter—group—project groups—aggregate*) consists of the following components:

- J and F are the same as for non-GROUP BY queries.
- G is similar to the P for non-GROUP BY queries. It is a sequence of functions $g_1, \ldots, g_r$, where $g_i : D[J] \to X_i$. The set X_i is arbitrary. Typically each g_i selects an attribute from $D[J]$. Let $X = X_1 \times \cdots \times X_r$.
- P_G is a sequence of functions $p_1, \ldots, p_{r'}$, where $p_i : X \to Y_i$. The set Y_i is arbitrary. Hence, if each g_i selected an attribute from $D[J]$ then each p_i is a function of those attributes.
- P_A is a sequence of functions $h_1, \ldots, h_s$, where $h_i : (D[J] \to \mathbb{N}) \to Z_i$. The set Z_i is arbitrary.

The application of a query $Q = (J, F, G, P_G, P_A)$ to a database $Y \in \mathcal{Y}$ proceeds as follows:

1. Let $A = \prod_{i=1}^{|J|} Y.J[i]$. Note that A is a multiset over $D[J]$.
2. Let $B \subseteq A$ be the multiset of all elements of A that satisfy F.
3. Let $C = \{(g_1(b), \ldots, g_r(b)) \mid b \in B\}$ be the set of *group keys*.
4. Let $D = \{(c, \{\!\{b \in B \mid (g_1(b), \ldots, g_r(b)) = c\}\!\}) \mid c \in C\}$ be the set of *groups*.
5. Output the multiset
   $\{\!\{(p_1(c), \ldots, p_{r'}(c), h_1(E), \ldots, h_s(E)) \mid (c, E) \in D\}\!\}$.

Assuming that the predicate F and the functions $g_1, \ldots, g_r$,
$p_1, \ldots, p_{r'}, h_1, \ldots, h_s$ can be specified in SQL, the query Q corresponds to the following SQL query:

$$\texttt{SELECT } \overline{[\![p_1]\!]}, \ldots, \overline{[\![p_{r'}]\!]}, \overline{[\![h_1]\!]}, \ldots, \overline{[\![h_s]\!]} \texttt{ FROM } J[1], \ldots, J[|J|] \texttt{ WHERE } \overline{[\![F]\!]} \texttt{ GROUP BY } \overline{[\![g_1]\!]}, \ldots, \overline{[\![g_r]\!]} \ .$$

Note that $\overline{[\![h_i]\!]}$ are the aggregations of attributes, e.g. `SUM`, `COUNT`, `MAX` etc.

3.3.13.3 Queries Combined using Set Operations. If Q_1 and Q_2 are supported queries over set semantics then we can also support queries $Q_1 \cup Q_2$, $Q_1 \cap Q_2$, and $Q_1 \setminus Q_2$. If the results of queries Q_1 and Q_2 are R_1 and R_2, respectively, then the results of the queries $Q_1 \cup Q_2$, $Q_1 \cap Q_2$, and $Q_1 \setminus Q_2$ are $R_1 \cup R_2$, $R_1 \cap R_2$, and $R_1 \setminus R_2$, respectively. Set operations can be combined any number of times but only at the top level, e.g. $Q_1 \setminus ((Q_2 \cup Q_3) \cap Q_4)$.

3.3.13.4 Sensitivity of queries without GROUP BY. Consider the queries described in Sec. 3.3.13.1. Note that we defined these queries to use multiset semantics, not set semantics.

In this analysis, we consider the "canonical" distance over databases, meaning that for datasets R, R' over the same schema we define their distance as the cardinality of their symmetric difference $(R \setminus R') \cup (R' \setminus R)$. For databases over the same schema we define their distance as the sum of the distances of the corresponding tables.

To find the sensitivity of a query $Q = (J, F, P)$ against a database schema $dbs = (t_1 : r_1, \ldots, t_m : r_m)$ with respect to the table t_i, we will find the *derivative* query $Q_{t_i}[y]$ of Q with respect to t_i. The derivative query is parameterized with a possible row y of the table t_i. The result of $Q_{t_i}[y]$ on a database Y is the multiset difference of the result of Q on Y and the result of Q on $Y[t_i \mapsto Y.t_i \setminus \{y\}]$ (database Y with one copy of row y removed from the table t_i). We will then find the maximum number of rows that may be returned by $Q_{t_i}[y]$ on Y, maximized over all possible y and Y. This is the sensitivity of Q with respect to the table t_i.

Let us first consider the case where $\mathsf{Dis}_{r_j} \neq \emptyset$ for all j. In this case the input tables cannot contain repeated rows, i.e. these multisets are actually sets. Due to the distributivity properties (Lemma 3.107) of the query $Q = (J, F, P)$, it is very simple to find its derivatives. Let $i_1, \ldots, i_k$ be the positions that contain t_i in J. Then

$$Q_{t_i}[y] = (J, F \wedge \bigvee_{j=1}^{k} F_{i_j}^{\{y\}}, P),$$

where $F_{i_j}^{\{y\}}$ is defined as in Lemma 3.107. Note that we used removals instead of additions to define the derivative query to ensure that the removed row y is included in the input of the derivative. Otherwise it would not be possible to express the derivative as a query in the sense of Sec. 3.3.13.1 (it would have to return a nonempty output on an empty input table).

Let $N \in \mathbb{N}$. The question of whether $Q_{t_i}[y]$ may return at least N rows can be phrased as a quantifier-free formula Φ in the manner we describe below. If this formula is satisfiable, then the answer to the query may contain at least N rows for some y. By varying N and checking for satisfiability of resulting formulas, we will find the value where it is no longer satisfiable. Satisfiability is modulo theories we commonly associate with the domains $D_1, \ldots, D_n$, where the attributes in the tables are picked from. Hence the theories may include Booleans, integers, reals and strings. The first three are well-supported by existing SMT solvers. We can also handle queries over string data, as long as the operations performed with them are supported by SMT solvers.

Note that because we are using multiset semantics, each combination of rows of input tables that satisfies F, produces exactly one output row, repeated output rows are not removed. Thus the formula will not depend on P.

The formula $\Phi_{Q,N}^{dbs,t_i}$ has three major parts: the variables, the functions (function variables), and the constraints. The conjunction of the constraints is the actual formula. Each variable and each function is described with its type, i.e. the set of values from which it can take values. The SMT solver tries to assign to each variable and function a value of its given type, such that the constraints become satisfied. The variables are the following:

- For each $n \in \{1, \ldots, N\}$, $j \in \{1, \ldots, |J|\}$, and $k \in \{1, \ldots, m\}$, where $r(a_1 : D_1, \ldots, a_m : D_m; \mathsf{Dis}_r)$ is the relation schema of the table $J[j]$: a variable $x[n, j, k]$ taking values in the set D_k. These variables enumerate the attributes of N rows from $D[J]$.
- For each $k \in \{1, \ldots, m\}$, where $r(a_1 : D_1, \ldots, a_m : D_m; \mathsf{Dis}_r)$ is the relation schema of the table t_i: a variable $y[k]$, taking values in the set D_k.

The functions are the following:

- For all $i \in \{1, \ldots, M\}$, where $dbs = (t_1 : r_1, \ldots, t_M : r_M)$, all minimal $I = \{i_1, \ldots, i_k\}$ such that $\{a_{i_1}, \ldots, a_{i_k}\} \in \mathsf{Dis}_r$, and all $\ell \in \{1, \ldots, m\} \setminus I$, where $r(a_1 : D_1, \ldots, a_m : D_m; \mathsf{Dis}_r)$ is the relation schema of the table t_i: a function $f[i, I, \ell] : D_{i_1} \times \cdots D_{i_k} \to D_\ell$.
- A function $g : D[J] \to \mathbb{Z}$.

The constraints of the formula, referring to the variables and functions above, are the following.

- For each $n \in \{1, \ldots, N\}$: the tuple of variables $x[n, \cdot, \cdot]$ must satisfy the predicate F.
- For each $n \in \{1, \ldots, N\}$, the following disjunction over all $j \in \{1, \ldots, |J|\}$, where $J[j] = t_i$, must hold:
  - Each element of the disjunction is a conjunction of the statements $y[k] = x[n, j, k]$ over all $k \in \{1, \ldots, m\}$, where $r(a_1 : D_1, \ldots, a_m : D_m; \mathsf{Dis}_r)$ is the relation schema of the table t_i.

- For all $i \in \{1, \ldots, M\}$, where $dbs = (t_1 : r_1, \ldots, t_M : r_M)$, for all minimal $I = \{i_1, \ldots, i_k\}$ such that $\{a_{i_1}, \ldots,$
  $a_{i_k}\} \in \mathsf{Dis}_r$, and all $\ell \in \{1, \ldots, m\} \backslash I$, where $r(a_1 : D_1, \ldots, a_m : D_m; \mathsf{Dis}_r)$ is the relation schema of the table t_i, for all $n \in \{1, \ldots, N\}$, $j \in \{1, \ldots, |J|\}$ with $J[j] = i$, the equality $x[n, j, \ell] = f[i, I, \ell](x[n, j, i_1], \ldots,$
  $x[n, j, i_k])$ must hold. Such equalities express the distinctness requirement in the relation schemas.
- The values $g(x[1, \cdot, \cdot]), \ldots, g(x[N, \cdot, \cdot])$ must be distinct, meaning that we actually get N different rows. Using the Z3 SMT solver, this constraint can be expressed through the `distinct`-predicate.

The discussions above have sketched the proof of the following lemma.

Lemma 3.108. *Let $dbs = (t_1 : r_1, \ldots, t_m : r_m)$ be a database schema such that $\mathsf{Dis}_{r_j} \neq \emptyset$ for all j, $Q = (J, F, P)$ a query over it, t_i a table in this schema and $N \in \mathbb{N}$. Then $\Phi_{Q,N}^{dbs,t_i}$ is satisfiable iff the sensitivity of Q with respect to the table t_i is at least N.*

Proof. Let $i_1, \ldots, i_k$ be the positions that contain t_i in J.

The *if* part. Suppose that the sensitivity of Q w.r.t. table t_i is at least N. Then there exists a database Y over dbs and a row $y \in D[t_i]$, such that $Q_{t_i}[y] = (J, F \wedge \bigvee_{j=1}^{k} F_{i_j}^{\{y\}}, P)$ returns at least N rows on Y. Each of those rows is obtained from an element of $D[J]$ satisfying $F \wedge \bigvee_{j=1}^{k} F_{i_j}^{\{y\}}$, using functions in P. These elements are all distinct because the input tables cannot contain repeated elements. Take N of these distinct elements of $D[J]$ and use them as valuations for the variables $x[1, \cdot, \cdot], \ldots, x[N, \cdot, \cdot]$. Then for all $n \in \{1, \ldots, N\}$, the tuple of variables $x[n, \cdot, \cdot]$ satisfies the predicate F. Thus the first constraint of $\Phi_{Q,N}^{dbs,t_i}$ is satisfied.

The function g can be defined as $g(x[n, \cdot, \cdot]) = n$ for all $n \in \{1, \ldots, N\}$ and arbitrarily on other arguments. The definition is correct because the tuples $x[n, \cdot, \cdot]$ are distinct for distinct n. Then also the fourth constraint of $\Phi_{Q,N}^{dbs,t_i}$ is satisfied.

Use the row y as the valuation of the variables $y[\cdot]$. Then for all $n \in \{1, \ldots, N\}$, the tuple $b = x[n, \cdot, \cdot]$ satisfies $F_{i_j}^{\{y\}}$ for at least one j. Then $b[i_j] \in \{y\}$, i.e. $x[n, i_j, \cdot] = y[\cdot]$. Thus the second constraint of $\Phi_{Q,N}^{dbs,t_i}$ is satisfied.

Suppose that $\{a_{j_1}, \ldots, a_{j_k}\} \in \mathsf{Dis}_r$, where $r(a_1 : D_1, \ldots, a_m : D_m; \mathsf{Dis}_r)$ is the relation schema of the table t_i. Then by Sec. 3.2.1.1, for each $(x_{j_1}, \ldots, x_{j_k}) \in D_{j_1} \times \cdots \times D_{j_k}$, there is at most one $(y_1, \ldots, y_m) \in D[t_i]$ satisfying $y_{j_1} = x_{j_1}, \ldots, y_{j_k} = x_{j_k}$. Let $I = \{j_1, \ldots, j_k\}$ and for all $\ell \in \{1, \ldots, m\} \setminus I$, let $f[i, I, \ell](x_{j_1}, \ldots, x_{j_k}) = y_\ell$ if y_ℓ exists and arbitrary otherwise. This gives a valuation of $f[\cdot, \cdot, \cdot]$ such that the third constraint of $\Phi_{Q,N}^{dbs,t_i}$ is satisfied.

The *only if* part. Suppose that there exists a valuation of $x[\cdot, \cdot, \cdot], y[\cdot], f[\cdot, \cdot, \cdot], g$ such that $\Phi_{Q,N}^{dbs,t_i}$ is satisfied. We construct a database Y such that for all $i \in \{1, \ldots, m\}$, $Y.t_i = \{x[n, j, \cdot] \mid n \in \{1, \ldots, N\}, j \in \{1, \ldots, |J|\}, J[j] = t_i\}$. Note that here we use the set constructor, i.e. repeated elements are removed. The third constraint of $\Phi_{Q,N}^{dbs,t_i}$ guarantees that Y is a database over dbs. Let $y = y[\cdot]$. The first two constraints of $\Phi_{Q,N}^{dbs,t_i}$ guarantee that for each $n \in \{1, \ldots, N\}$, the tuple $x[n, \cdot, \cdot]$ produces (using functions in P) a row in the result of $Q_{t_i}[y] = (J, F \wedge \bigvee_{j=1}^{k} F_{i_j}^{\{y\}}, P)$ on Y. The fourth constraint guarantees that those N rows are separate elements of the multiset (although some may be equal to each other). Thus $Q_{t_i}[y]$ returns at least N rows on Y, hence the sensitivity of Q w.r.t. table t_i is at least N. □

3.3.13.5 Repeated Rows. Now consider the case where the input tables may contain repeated rows. We prove the following generalization of Lemma 3.108.

Theorem 3.109. *Let $dbs = (t_1 : r_1, \ldots, t_m : r_m)$ be a database schema, Q a query over it, t_i a table in this schema (w.r.t. which we want to estimate the sensitivity of Q) and $N \in \mathbb{N}$. Then there are four cases:*
(1) $\mathsf{Dis}_{r_j} \neq \emptyset$ *for all* j
(2) $\mathsf{Dis}_{r_j} \neq \emptyset$ *for all* $j \neq i$, $\mathsf{Dis}_{r_i} = \emptyset$, *and* t_i *occurs only once in* J
(3) $\mathsf{Dis}_{r_j} \neq \emptyset$ *for all* $j \neq i$, $\mathsf{Dis}_{r_i} = \emptyset$, *and* t_i *occurs more than once in* J

(4) $\mathsf{Dis}_{r_j} = \emptyset$ *for some* $j \neq i$

In cases (1) and (2), $\Phi_{Q,N}^{dbs,t_i}$ *is satisfiable iff the sensitivity of* Q *with respect to the table* t_i *is at least* N*. In cases (3) and (4),* $\Phi_{Q,N}^{dbs,t_i}$ *is satisfiable iff the sensitivity of* Q *with respect to the table* t_i *is* ∞ *and not satisfiable iff the sensitivity of* Q *with respect to the table* t_i *is 0.*

Proof. We first consider the sensitivity of Q (w.r.t. t_i) over the database schema $dbs' = (t_1 : r'_1, \ldots, t_m : r'_m)$ where $r'_j = r_j(a_1 : D_1, \ldots, a_m : D_m; \mathsf{Dis}_{r_j})$ if $\mathsf{Dis}_{r_j} \neq \emptyset$ and $r'_j = r'_j(a_1 : D_1, \ldots, a_m : D_m; \{\{a_1, \ldots, a_m\}\})$ if $\mathsf{Dis}_{r_j} = \emptyset$. The schema dbs' forbids repeated rows but is otherwise the same as dbs. If the sensitivity of Q over dbs' is 0 then the predicate F is false for all possible inputs and thus the result of the query Q (over both dbs' and dbs) is empty for all possible inputs. Thus the sensitivity of Q over dbs is also 0.

Now suppose that the sensitivity of Q over dbs' is at least 1. Then there exists a database Y over dbs' and a row $y \in \mathcal{X}_{r_i}$ such that adding y to $Y.t_i$ adds at least 1 row to the result of Q. Let c be a one of the rows added to the result and $b \in D[J]$ the combination of rows of input tables that produces c.

If $\mathsf{Dis}_{r_j} = \emptyset$ for some $j \neq i$ then we can let $Y.t_j$ contain $m \in \mathbb{N}$ copies of $b[\ell]$ where $J[\ell] = t_j$, which will cause adding y to $Y.t_i$ to produce at least m added copies of c in the result. Because m may be arbitrarily large, the sensitivity of Q over dbs is infinite.

If $\mathsf{Dis}_{r_i} = \emptyset$ and t_i occurs more than once in J (i.e. $k \geq 2$, where $i_1, \ldots, i_k$ are the positions that contain t_i in J) then at least one of $b[i_1], \ldots, b[i_k]$ must be equal to y. Let ℓ be such that $b[i_\ell] = y$. Let $s \in \{i_1, \ldots, i_k\} \setminus \{i_\ell\}$. Then we can let $Y.t_i$ contain $m \in \mathbb{N}$ copies of $b[s]$ (which may or may not be equal to y), which will cause adding y to $Y.t_i$ to produce at least m added copies of c in the result. Because m may be arbitrarily large, the sensitivity of Q over dbs is infinite.

If $\mathsf{Dis}_{r_j} \neq \emptyset$ for all $j \neq i$, $\mathsf{Dis}_{r_i} = \emptyset$, and t_i occurs only once in J then we can use part (a) of Lemma 3.107 to get that the result of $Q_{t_i}[y]$ over dbs on database Y where $y \in Y.t_i$, is equal to the result of Q over dbs on database $Y' = Y[t_i \mapsto \{y\}]$. The database Y' does not contain repeated rows in any table, thus the result is the same over dbs' and is also equal to the result of $Q_{t_i}[y]$ over dbs'. Thus the sensitivity of Q (w.r.t. t_i) over dbs is equal to the sensitivity of Q over dbs'. □

3.3.13.6 Tables Used More than Once. The description of the computation of the sensitivity of a query in Sec. 3.3.13.4 allows using an input table more than once in the join. This is not the only way this feature can be implemented in the analyzer. Instead, the analyzer can compute the sensitivities of the query as if all the tables were different. Then it can add the sensitivities w.r.t. each copy of the table together to obtain the sensitivity w.r.t. the original table, similarly to the partial derivatives of compound multi-variable functions.

This implementation choice has the advantage that because the sensitivities w.r.t. each copy of a table may be lower than w.r.t. the original table, the SMT solver is more likely to terminate within reasonable time (see Sec. 4.3.6).

The disadvantage is that this version of the analysis is not as precise. The actual sensitivity may be lower than the obtained result (it cannot be higher because the computed sensitivities are global, i.e. they do not depend on the actual value of the input database).

3.3.13.7 Sensitivity of Queries with Set Semantics. We define the sensitivity of a query Q over set semantics w.r.t. table t_i as the maximum number of rows in the derivative $Q_{t_i}\{y\}(Y)$, maximized over all possible y and Y, where $Q_{t_i}\{y\}(Y)$ is the set difference of the result of Q on Y and the result of Q on $Y[t_i \mapsto Y.t_i \setminus \{y\}]$ (database Y with the row y (if it exists) removed from the table t_i).

To find the sensitivity of a query in set semantics (SELECT DISTINCT in SQL) we modify the formula $\Phi_{Q,N}^{dbs,t_i}$ as follows. Let the modified formula be $\Psi_{Q,N}^{dbs,t_i}$. In set semantics, repeated rows in the output are removed. Thus the formula will now depend on P. Instead of requiring the distinctness of the combinations of rows of input tables, we require the distinctness of the rows in the answer of the query. The following variables are added:

- For each $n \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, r\}$, where r is the length of P: a variable $z[n, k]$, taking values in the set X_k, which is the range of the mapping p_k.

The following constraints are added:

- For each $n \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, r\}$: the equality $z[n, k] = p_k(x[n, \cdot, \cdot])$ must hold, where p_k is the k-th mapping in P. These constraints state that the outputs of the query are computed according to P.

The function g is replaced with the following:

- Function $g : X_1 \times \cdots \times X_r \to \mathbb{Z}$, where X_i is the range of the mapping p_i.

The distinctness constraint is replaced:

- The values $g(z[1, 1], \ldots, z[1, r]), \ldots, g(z[N, 1], \ldots,$ $z[N, r])$ must be distinct. This expresses the distinctness of the rows in the answer to the query.

In set semantics, we get the following weaker analogue of Theorem 3.109 that sometimes holds only in one direction.

Theorem 3.110. *Let $dbs = (t_1 : r_1, \ldots, t_m : r_m)$ be a database schema, Q a query over it, t_i a table in this schema and $N \in \mathbb{N}$. **(a)** If the sensitivity of Q (with set semantics) with respect to table t_i is at least N, then $\Psi_{Q,N}^{dbs,t_i}$ is satisfiable. **(b)** If t_i occurs only once in J and $\Psi_{Q,N}^{dbs,t_i}$ is satisfiable, then the sensitivity of Q with respect to the table t_i is at least N.*

Proof. Let $i_1, \ldots, i_k$ be the positions that contain t_i in J.

Part *(a)*. Suppose that the sensitivity of $Q = (J, F, P)$ w.r.t. table t_i is at least N. Then there exists a database Y over *dbs* and a row $y \in D[t_i]$, such that $R \setminus R_1$ where R is the result (over set semantics) of Q on Y and R_1 is the result (over set semantics) of Q on $Y[t_i \mapsto Y.t_i \setminus \{y\}]$, contains at least N rows. Let R' and R'_1 be the results of Q on Y and $Y[t_i \mapsto Y.t_i \setminus \{y\}]$, respectively, *over multiset semantics*. Then $R \setminus R_1$ contains a row r iff R' contains r at least once and R'_1 does not contain r. Because $R \setminus R_1$ is a set, its elements (rows) are distinct. It contains at least N distinct rows. Consider one of those rows r. Then $R' \setminus R'_1$ (the result of $Q_{t_i}[y] = (J, F \wedge \bigvee_{j=1}^{k} F_{i_j}^{\{y\}}, P)$ on Y over multiset semantics) contains r at least once. Thus $Q_{t_i}[y]$ returns at least N *distinct* rows on Y. Each of those distinct rows is obtained from an element of $D[J]$ satisfying $F \wedge \bigvee_{j=1}^{k} F_{i_j}^{\{y\}}$, using functions in P. Take N of these elements of $D[J]$ and use them as valuations for the variables $x[1, \cdot, \cdot], \ldots, x[N, \cdot, \cdot]$. Then for all $n \in \{1, \ldots, N\}$, the tuple of variables $x[n, \cdot, \cdot]$ satisfies the predicate F. Thus the first constraint of $\Phi_{Q,N}^{dbs,t_i}$ (also a constraint of $\Psi_{Q,N}^{dbs,t_i}$) is satisfied.

Take the N elements of $R \setminus R_1$ obtained (using functions in P) from the ones taken from $D[J]$, and use as valuations for the variables $z[\cdot, \cdot]$. Then also the constraint added to $\Psi_{Q,N}^{dbs,t_i}$ is satisfied.

The function g can be defined as $g(z[n, \cdot]) = n$ for all $n \in \{1, \ldots, N\}$ and arbitrarily on other arguments. The definition is correct because the tuples $z[n, \cdot]$ are distinct for distinct n. Then also the distinctness constraint of $\Psi_{Q,N}^{dbs,t_i}$ is satisfied.

The second and the third constraint of $\Phi_{Q,N}^{dbs,t_i}$ are handled the same as in the proof of Lemma 3.108 (last two paragraphs of the if part).

Part *(b)*. Suppose that t_i occurs only once in J and that there exists a valuation of $x[\cdot, \cdot, \cdot], y[\cdot], z[\cdot, \cdot], f[\cdot, \cdot, \cdot], g$ such that $\Psi_{Q,N}^{dbs,t_i}$ is satisfied. We construct a database Y such that for all $i \in \{1, \ldots, m\}$, $Y.t_i = \{x[n, j, \cdot] \mid n \in \{1, \ldots, N\}, j \in \{1, \ldots, |J|\}, J[j] = t_i\}$. Note that here we use the set constructor, i.e. repeated elements are removed. The third constraint of $\Phi_{Q,N}^{dbs,t_i}$ guarantees that Y is a database over *dbs*. Let $y = y[\cdot]$. The first two constraints of $\Phi_{Q,N}^{dbs,t_i}$ and the added constraint of $\Psi_{Q,N}^{dbs,t_i}$ guarantee that for each $n \in \{1, \ldots, N\}$, the tuple $z[n, \cdot]$ is a row in the result of $Q_{t_i}[y] = (J, F \wedge \bigvee_{j=1}^{k} F_{i_j}^{\{y\}}, P)$ on Y. Because t_i occurs only once in J, we have $k = 1$ and $\bigvee_{j=1}^{k} F_{i_j}^{\{y\}} = F_{i_1}^{\{y\}}$. This predicate is equivalent to using from the table t_i only the element y. Thus $z[n, \cdot]$ is a row in the result of Q on $Y[t_i \mapsto \{y\}]$, which is equal to the set difference of the result of Q on $Y[t_i \mapsto \{y\}]$ and the (empty) result of Q on $Y[t_i \mapsto \emptyset]$. The distinctness constraint of $\Psi_{Q,N}^{dbs,t_i}$ guarantees that those N rows are distinct. Hence the sensitivity of Q w.r.t. table t_i is at least N. □

If t_i occurs many times in J, then the satisfiability of $\Psi^{dbs,t_i}_{Q,N}$ does not imply the sensitivity N of Q, because some of the N rows that would be added to the output when adding one row r to the table t_i, could still be constructible if all rows not used in the construction of those N rows, and also r, are removed from the database. In this case, the number of added rows will be less than N.

3.3.13.8 Sensitivity of GROUP BY Queries. To find the sensitivity of a query $Q = (J, F, G, P_G, P_A)$ against a database schema $dbs = (t_1 : r_1, \ldots, t_m : r_m)$ with respect to the table t_i, we will still find the derivative query $Q_{t_i}[y]$ of Q with respect to t_i. The result of $Q_{t_i}[y]$ on a database Y is the multiset difference $R_1 \setminus R_2$ where R_1 is the result of Q on Y and R_2 is the result of Q on $Y[t_i \mapsto Y.t_i \setminus \{y\}]$ (database Y with one copy of row y removed from the table t_i). Unlike in Sec. 3.3.13.4, R_2 is not necessarily a subset of R_1 because removing y from Y may not only remove groups but also change them. The set $R_1 \setminus R_2$ contains both the removed and the changed groups, hence we can still measure its size to find the sensitivity of Q.

We will then find the maximum number of rows that may be returned by $Q_{t_i}[y]$, maximized over all possible y. This is the sensitivity of Q with respect to the table t_i.

To find an upper bound on the sensitivity of Q, we find an upper bound on the sensitivity of the query $Q' = (J, F, G)$ considered as a non-GROUP BY query, over set semantics because the same group cannot be used in multiple rows of the output. Note that the formula in Sec. 3.3.13.7 considers N rows in the output that can be constructed using a row r after r is added to an input table. Some of these N rows (let this subset be S) may have been constructible even before r was added to the input table. To get a precise result, the rows in S should have been excluded but we could not express this in an SMT formula. For GROUP BY queries, we must actually *not* exclude the rows in S because these represent the keys of groups that were already in the result but are potentially modified when r is added. The rest of the N rows represent the keys of new groups created by the addition of r. Thus we must use the method in Sec. 3.3.13.7 to find this upper bound instead of the exact sensitivity of Q'. Then this is an upper bound on the sensitivity of Q.

We get the following theorem:

Theorem 3.111. *Let $dbs = (t_1 : r_1, \ldots, t_m : r_m)$ be a database schema, $Q = (J, F, G, P_G, P_A)$ a GROUP BY query over it, t_i a table in this schema and $N \in \mathbb{N}$. Let $Q' = (J, F, G)$. Then there are two cases:*
(1) t_i occurs only once in J
(2) t_i occurs more than once in J
In case (1), $\Psi^{dbs,t_i}_{Q',N}$ is satisfiable iff the sensitivity of Q with respect to the table t_i is at least N. In case (2), if the sensitivity of Q over set semantics with respect to the table t_i is at least N then $\Psi^{dbs,t_i}_{Q',N}$ is satisfiable.

Proof. **Case (1).** The result of the derivative $Q'_{t_i}[y]$ on the database $Y[t_i \mapsto \{y\}]$ contains the keys of the groups added to the result of Q when $Y[t_i \mapsto \emptyset]$ is replaced with $Y[t_i \mapsto \{y\}]$. These are also the keys of the groups that are potentially changed in or added to the result of Q when Y is replaced with $Y[t_i \mapsto Y.t_i \cup \{y\}]$. The number of these keys is an upper bound on the number of rows in $Q_{t_i}\{y\}(Y[t_i \mapsto Y.t_i \cup \{y\}])$. This upper bound is achievable, e.g. when $Y.t_i = \emptyset$. Thus the sensitivity of Q w.r.t. t_i is the maximum number of rows in the derivative $Q'_{t_i}\{y\}(Y'')$ over databases Y'' where $Y''.t_i = \{y\}$. Because for all Y'' for which $y \in Y''.t_i$, the derivative $Q'_{t_i}\{y\}(Y'')$ is monotonically decreasing in $Y''.t_i$, and for all Y'' for which $y \notin Y''.t_i$, the derivative $Q'_{t_i}\{y\}(Y'')$ is empty, the sensitivity of Q w.r.t. t_i is the maximum number of rows in the derivative $Q'_{t_i}\{y\}(Y'')$ over all databases Y'', i.e. the sensitivity of Q' over set semantics w.r.t. t_i.

Thus the sensitivity of Q w.r.t. table t_i is at least N iff the sensitivity of Q' w.r.t. t_i (over set semantics) is at least N. By Theorem 3.110, the sensitivity of Q' w.r.t. t_i (over set semantics) is at least N iff $\Psi^{dbs,t_i}_{Q',N}$ is satisfiable.

Case (2). Suppose that the sensitivity of Q over set semantics with respect to the table t_i is at least N but $\Psi^{dbs,t_i}_{Q',N}$ is not satisfiable. By Theorem 3.110, then the sensitivity of Q' (with set semantics) w.r.t. table t_i is less than N. From the discussion before the statement of the current theorem (Theorem 3.111), we get that this upper bound also applies to the sensitivity of Q, i.e. the sensitivity of Q over set semantics w.r.t. the table t_i is less than N. Contradiction. □

Aggregation queries. An aggregation query can be seen as a GROUP BY query $Q = (J, F, G, P_G, P_A)$ with $G = \emptyset$ and $P_G = \emptyset$. These always return exactly 1 row, thus 1 can be taken as the upper bound on the sensitivity. Only if the query with the aggregation operators removed (i.e. replacing SUM(e) with e, etc.) has sensitivity 0 then also the aggregating query has sensitivity 0.

3.3.13.9 Finding the Sensitivity of a Workflow. Let *dbs* be a database schema and $\langle t_i : r_i = Q_i \rangle_{i=1}^{k}; Q_{k+1}$ a workflow over it. Let t be a table in *dbs*. We find an upper bound on the sensitivity of the workflow w.r.t. the table t similarly to the analysis presented in Sec. 3.3.2, using the composition properties of sensitivity. The computation of sensitivities considers each query $Q_1, \ldots, Q_{k+1}$ one by one. Namely,

- suppose that we have already found upper bounds c_j, where $1 \le j \le k$, for the sensitivity of the workflow $\langle t_i : r_i = Q_i \rangle_{i=1}^{j-1}; Q_j$ with respect to the table t;
- let c' be an upper bound on the sensitivity of the query Q_{k+1} with respect to the table t, and c'_i an upper bound on the sensitivity of the query Q_{k+1} with respect to the table t_i.
  - Note that Q_{k+1} is a query over $dbs \cup \{t_1 \mapsto r_1, \ldots, t_k \mapsto r_k\}$.

Then $c' + \sum_{i=1}^{k} c_i c'_i$ is an upper bound on the sensitivity of $\langle t_i : r_i = Q_i \rangle_{i=1}^{k}; Q_{k+1}$ w.r.t. t. This statement follows directly from Prop. 3.2.

Set operations. The sensitivity of the combinations of queries with set operations is found by making this set operation an element of the workflow. If $\otimes$ is a set operation (union, intersection, or difference), then we rewrite the query $Q_1 \otimes Q_2$ as $\langle t_1 : r = Q_1, t_2 : r = Q_2 \rangle t_1 \otimes t_2$, where r is the schema of the output of both Q_1 and Q_2. We find the sensitivity of this workflow w.r.t. the input table we are interested in, considering that the sensitivity of $t_1 \otimes t_2$ is 1 w.r.t. both t_1 and t_2. Such rewriting is also applicable inside a larger workflow.

3.3.13.10 Finding Uniqueness Constraints on the Result Table. The schema $r(a_1 : D_1, \ldots, a_n : D_n; \mathsf{Dis}_r)$ of each input table (relation) includes the set of uniqueness constraints $\mathsf{Dis}_r \subseteq \mathcal{P}(\{a_1, \ldots, a_n\})$ on the relation r. When analyzing a query, we would also like to determine such constraints on the output table of the query, in order to use them at the next steps of the workflow.

Let $r(a_1 : D_1, \ldots, a_n : D_n; \mathsf{Dis}_r)$ now denote the schema of the output relation. Suppose we want to check whether $\{a_{i_1}, \ldots, a_{i_k}\} \in \mathsf{Dis}_r$. This can be checked similarly to determining the sensitivity of the query. Depending on whether a query Q removes repeated rows from the output or not, we use either the formula in Sec. 3.3.13.7 or Sec. 3.3.13.4, respectively, as the base formula. This formula allows us to check whether Q can return at least N rows with the restriction that for each component corresponding to a certain input table t_i, this component must be the same for all N rows of the joined and filtered table used to produce the N output rows. This restriction is encoded in the formula using the variables $y[k]$ and the constraints on these variables.

Currently, we want to check whether Q can return at least $N = 2$ rows with the restriction that for each component in the set $\{a_{i_1}, \ldots, a_{i_k}\}$, this component is the same for all N output rows. This restriction can be encoded by replacing the variables $y[k]$ in the previous formula with the following:

- For each $j \in \{1, \ldots, k\}$, where $r(a_1 : D_1, \ldots, a_m : D_m; \mathsf{Dis}_r)$ is the relation schema of the output table: a variable $y[j]$, taking values in the set D_{i_j}. Note that Dis_r is not yet known at this point but we are not using it here. The rest of the schema of r can be determined using type derivation.

The constraints on these variables are replaced with the following:

- For each $n \in \{1, \ldots, N\}$, the following conjunction must hold:
  - a conjunction of the statements $y[j] = z[n, i_j]$ over all $j \in \{1, \ldots, k\}$.

These constraints use the z variables, which were defined in Sec. 3.3.13.7. If we used the formula from Sec. 3.3.13.4 as the base formula, we must add the z variables and the constraints on them (except the

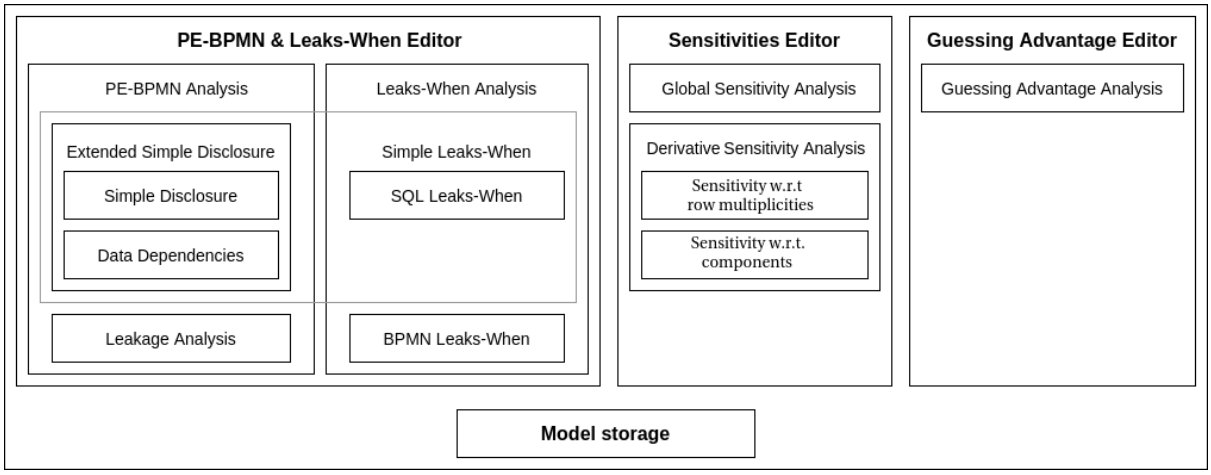


Figure 47: Pleak Architecture

one distinctness constraint) from Sec. 3.3.13.7 to the formula. The resulting formula is satisfiable iff $\{a_{i_1}, \ldots, a_{i_k}\} \notin \mathsf{Dis}_r$.

For GROUP BY queries $Q = (J, F, G, P_G, P_A)$, if P_G selects the same expressions as G (i.e. if G is a sequence of functions $g_1, \ldots, g_r$ and P_G is a sequence of functions $p_1, \ldots, p_r$ such that $p_i(x_1, \ldots, x_r) = x_i$) then we can compute uniqueness constraints for the query $Q' = (J, F, G)$. These are also uniqueness constraints for the query Q because these determine which sets of attributes uniquely determine the group key and the group key uniquely determines the whole result row.

For queries combined using set operations, we combine uniqueness constraints as follows:

- $\{a_{i_1}, \ldots, a_{i_k}\}$ is a uniqueness constraint for $Q_1 \cap Q_2$ if it is a uniqueness constraint for Q_1 or Q_2.
- $\{a_{i_1}, \ldots, a_{i_k}\}$ is a uniqueness constraint for $Q_1 \setminus Q_2$ if it is a uniqueness constraint for Q_1.
- We cannot derive any uniqueness constraints for $Q_1 \cup Q_2$ from those for Q_1 and Q_2 because even if there is at most one $(y_1, \ldots, y_n) \in R$ satisfying $y_{i_1} = x_{i_1}, \ldots, y_{i_k} = x_{i_k}$ in the result of Q_1 and at most one in the result of Q_2, the ones are not necessarily equal, thus there may be two such rows in the result of $Q_1 \cup Q_2$.

3.4 Pleak Architecture

Pleak [68] is a web-based tool[11] to give an interface to all of the analysis designed in the NAPLES project. The Pleak tool contains three editors that serve as an interface to respective analyzers as illustrated in Figure 47[12]. These editors combine the analysis that take the common inputs or they are integrated together to support each other. The editors are used to define the inputs needed for the analysis tools (e.g., the PE-BPMN editor supports adding of the stereotypes and other editors allow to specify the SQL scripts for these stereotyped tasks).

The *PE-BPMN & Leaks-When Editor* are joint because Leaks-When analysis has some support for the stereotypes defined in the PE-BPMN editor. For example, the extended simple disclosure analysis can be used to simplify leaks-when output in order to get Simple Leaks-When results. In addition both analyzers work with collaboration models.

The *Sensitivities Editor* is focused on working with the data processing workflows that have aggregation queries. Moreover, this editor allows user to input data tables that are not considered in the PE-BPMN or Leaks-When analysis.

The *Guessing Advantage Editor* is similar to Sensitivities Editor, but different analysis interface. While the data added to the model remains the same (as in the Sensitivities Editor), the analysis itself

[11] running in `pleak.io`

[12] This structure is also reflected in our source code in `https://github.com/pleak-tools`

takes different input parameters. Moreover, Guessing Advantage Editor also supports collaboration models in addition to the data processing workflows.

Pleak uses the *Model storage* for storing, managing, and sharing model information. It stores the model and associated metadata of the model as well as provides an interface to work with the models. For instance, the model contents (in the XML format) is stored in the files. The meta information (such as author, timestamp and similar, and it is stored as records in the database.

3.5 Secure Multiparty Computation in NAPLES

3.5.1 Work Approach. Taking into consideration the goals of the NAPLES Extension and Cyber PA projects it was decided that the best approach to benefit the Cyber PA project and achieve NAPLES Extension goals (described in Sec. 2.5) would be to:

Test information sharing and processing in a privacy-preserving environment where the information shared is not seen to the stakeholders, the processing of data is done in an encrypted form and the output of the processing contains valuable information for all contributors without revealing confidential information.

On a high-level the work in this track can be divided into two bigger work items:

1. At first analysis was carried out in order to detail the exact use case being worked on and record the preliminary design of the prototype solution into an analysis document. Pleak technology and capabilities were used as a part of analysis and documenting.
2. Followed by prototype implementation, which is based on the preliminary analysis. Prototype implementaton specifics and an overview of the prototype usefulness for the Cyber PA project were then included in the analysis documentation created in first step.

The following chapters will provide an overview of the two high level work items.

3.5.2 Solution Analysis and Design. Currently most event logs are text messages intended for humans to read. Because of the programming patterns behind event log generation, log messages tend to have implicit structure which can be parsed using natural language processing parsing methods, like regular expressions etc. Implicit structure can be extracted manually by writing respective grammar by human or automatically using some pattern recognition algorithm. State of the art for now is writing those patterns manually. Log entries are parsed to structured form, consisting of set of key-value pairs. Keys are explicitly defined with pattern matching language and values are filled in during pattern matching. When event log entries are parsed to structured form, it is possible to find correlations between different log entries. For this work we defined correlation as a fact that two parsed log entries have at least one pair of key-value pairs that share the same value.

However, the problem of logs correlation is not one that has an easy solution. If a generic all-purpose system was possible it would have already been created and the problem of logs correlation would already have been solved.

- System logs are mainly recorded in some human readable form, requiring text processing. The format of different system logs may differ completely. Meaning a successful solution should be devised so that input data processing is as flexible as possible and new logs can be incorporated easily without needing extensive configurations to the solution.
- System logs hold a lot of sensitive information. Not only the log contents, but also the information on what logs and where are being generated from. In order to enable the combining and processing of logs from multiple mistrusting parties it is crucial the parties are not able to learn each other's secrets.
- At least to some extent, any approach would still require maintenance and configurations. As systems and logs will evolve over time any correlation approach will need to keep up with these changes. Additionally, new technologies and advancements may result in new types of events the correlation system should be looking for.

- The huge amount of log entries for processing has both positive and negative effects. With more entries the chances that suspicious activities are included in the dataset being processed are higher. Additionally, combining logs from different sources may help identify patterns that would not be visible otherwise. On the other side, the increased number of logs involved may also obscure that suspicious activity has taken place.
- Complexity of the analysis carried out and the amount of data requires a lot of processing power and therefore takes time to complete. If the results of the correlation are meant to be used to counter active attacks then processing should be as fast as possible. Even though logs correlation also allows for root cause analysis, the main benefit of such a system would still be the capability to respond to ongoing attacks and minimize impact. There is a fine line between how detailed the correlation can be, how fast results are achieved and how trustworthy are the identified threat situation.

For the purpose of this project it was decided to prototype a privacy preserving log correlation engine for several mistrusting parties using the approaches and considering the problems described above. We will correlate the event logs in privacy-preserving environment to detect joint events taking place. This gives organizations ability to make clear distinction between mundane, possibly confidential every-day events and unusual, possibly harmful events which should be analyzed further. This methodology should be useful even without following information sharing and joint analysis, therefore it should increase the will and likelihood for enterprises to join the Cyber PA information gathering network.

The selected use case and approach allows to validate the Pleak and Sharemind technologies and provides a useful prototype, together with learings, to the Cyber PA project, where the NAPLES Extension work can be incorporated to the final solution, if suitable.

3.5.3 Prototype Implementation. The goal of the prototype implementation of the event logs correlation use case is to:

- experiment with privacy-preserving event correlation;
- hand over the prototype implementation (together with analysis) to the Cyber PA project for enhancements and potential incorporation to Cyber PA Solution;
- provide preliminary overview of prototype implementation related learnings and assessment of suitability to Cyber PA framework (within analysis document).

Main requirements considered for the log correlations approach and prototype implementation were:

- Format for log normalization should be as simple as possible. As the use case assumes collaboration from several mistrusting parties, we assume that efforts to minimize required collaboration tasks, will pay off as increased will to collaborate at all. Normalization approach chosen was an unordered set of key-value pairs, where the key is statically named in the pattern matching rule and value is filled with that rule.
- The event correlation algorithm:
  - needs to be plain and obvious for auditing purposes (without significant training in mathematics and informatics).
  - should run both conventionally and in privacy-preserving settings without modifications.
  - should be fast and simple to ensure sufficient performance in privacy-preserving settings.

Due to strict restrictions on using real-life data it was decided that as a part of this project log parsing will not be carried out and a test data generator will be implemented, that will generate an already structured dummy dataset for the event correlation algorithm. In essence the data generator is a synthetic generator for random chains of structured log entries of random timestamps, keys and values. Parameters for the test data generator (including the random sets of timestamps, keys and values etc) were chosen so that the probabilistic outputs provide a wide coverage for testing the correlation algorithm.

The event correlation algorithm was first implemented in Python (non privacy-preserving) and then ported to Sharemind MPC (privacy-preserving version). This aligns with the processes proposed as a part of the use case analysis and approach.

4 RESULTS AND DISCUSSIONS

4.1 PE-BPMN

Support for PE-BPMN stereotypes and analysis has been implemented in JavaScript and is running in the browser. Hence we are not discussing the performance, instead this section focuses on an example of PE-BPMN model and analysis. We have demonstrated that the analysis is feasible for quite complex models, which is unsurprising because the used graph traversal methods are quite efficient. In addition to example scenarios, we discuss possible future directions for PE-BPMN notation and analysis. To showcase PE-BPMN analysis we use two scenarios from Mobile CRT. The first is using the general MPC stereotype while the second shows which additional details can be seen in the analysis if instead of the generic stereotypes we use the concrete function and additive secret sharing methods for secure computation and include all their details in the process as well as in the analysis.

4.1.1 Applying PE-BPMN Analysis to RapidGather. Mobile CRT creates an application called RapidGather [69] to use mobile phones to detect and respond to crisis situations. In addition, they are exploring other uses where the private data from mobile phones can be used to the benefit of the mobile device user such as agreeing on a meeting location without disclosing your current location or finding a free time in peoples calendar.

4.1.1.1 Stakeholders and Components. The stakeholders in the various processes for Mobile CRT either repeat or have close resemblances between the processes. Hence, this section summarizes the main stakeholders and serves as an introduction to the concrete processes in the following sections.

Mobile Device. The mobile device is an end-user device with common hardware and capabilities but running PE Android (privacy enhanced Android) [69]. PE Android is a Mobile CRT specific operating system that manages data access with an architecture that supports PAL modules to that manage data protection independently of the application that requests the data.

The main components of a mobile device are RapidGather application, PE Android layer and the PAL modules specific to different input data and privacy enhancign technologies. For example, SplitLocation PAL module takes a mobile device location as an input and produces two secret shares for further processing using PULSAR technologies.

Command Center. Command Center is the interface for law enforcement to use the data collected from the phones. More specifically, it has a query interface that allows to make the queries and then technical components to handle these queries with respect to the privacy technologies needed to process the queries. More concretely, the interface allows access to the query server that manages the correct way to perform the queries. Similarly to the PAL modules in the phone it has plugins for various privacy technologies and concrete setups to send the queries and receive the answers and apply privacy technologies as necessary for these actions.

Technology Specific Servers. Different privacy enhancing technologies or approaches use their own servers. For example, PULSAR servers to manage secure multiparty computation or PRESNA servers to do crisis detection. In some cases these servers could be maintained by the command center, but the

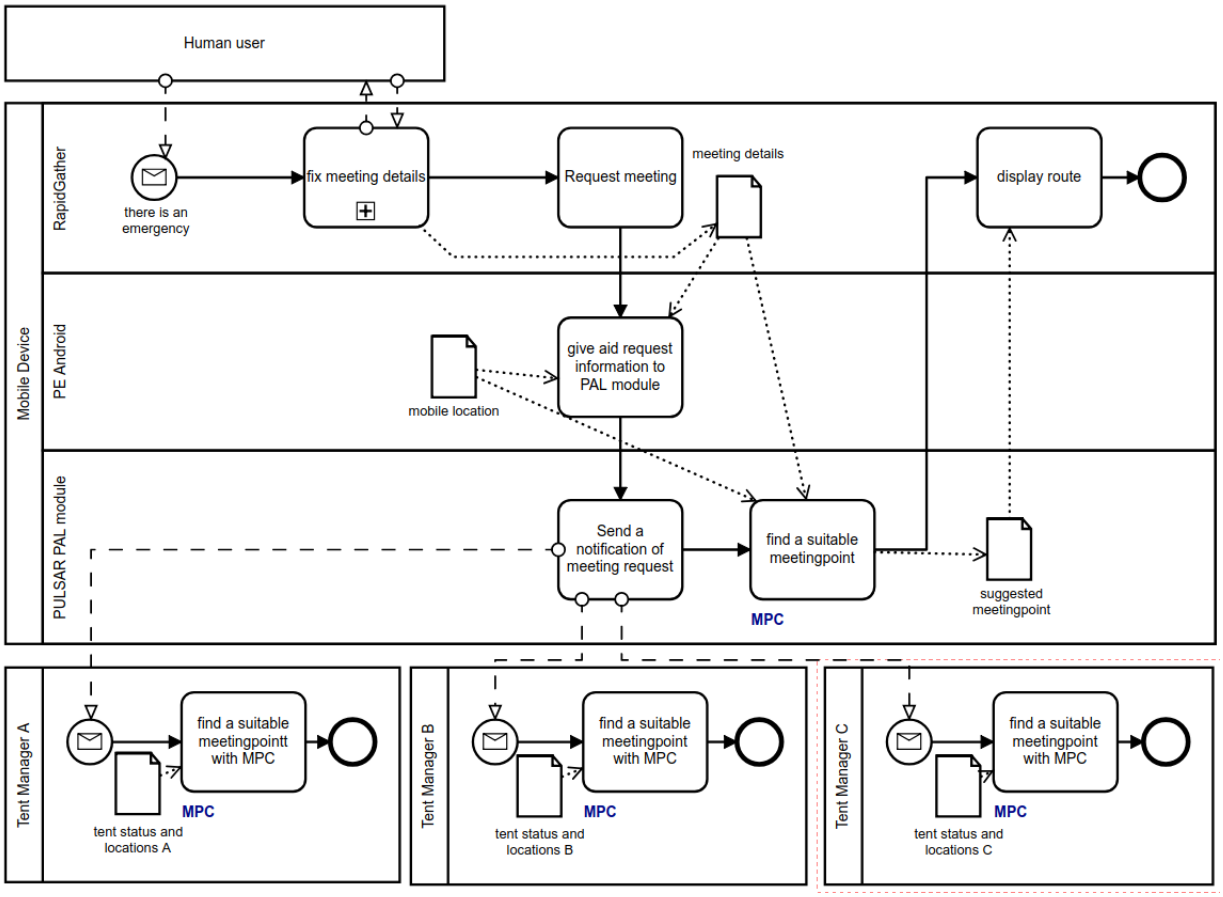


Figure 48: Rendezvous Scenario with MPC Stereotype in PE-BPMN

PULSAR servers for SMC need to be hosted by independent stakeholders. In addition, Helio servers require Intel SGX enabled processors.

4.1.1.2 Aid Tent and Rendezvous. Aid tent and Rendezvous scenarios are quite similar to each other and therefore this document considers them together. In short, the goal of both is to find a closest meetingpoint. For aid tent a mobile device user wants to find a suitable location for emergency aid and there is a manager of aid tents that knows which tents are available at any time. In rendezvous there could be multiple entities managing the potential meeting locations, for example the mobile devices correspond to agents on the field and the managers are different law enforcement and rescue agencies that are collaborating on a mission.

Analysis. Both scenarios are built using secure multiparty computation to find the meetingpoint without disclosing any additional information to either parties. The private data is the location of the mobile device user and the overall state of the potential meetingpoints. In addition, the underlying assumptions are that the parties performing secure computation are not collaborating, however, this is easily satisfied as all parties with any private data in these scenarios are participating in the computations.

The rendezvous scenario is a multiparty scenario and can be seen in Fig. 48 and `https://pleak.io/app/#/view/qoWJrlU769B9p-JV4vIl`. Aid tent is almost the same, but only has two parties doing the computations as seen in `https://pleak.io/app/#/view/AzdL8qKe_jxT_2JnegUi`. Here we also only give the simple disclosure report (see Table 11) and simple data dependency information (see Table 12) about the rendezvous variation. In addition, we demonstrate how these are joined in the extended disclosure table in Table 13. The aid tent case is very similar, but only has one data object for the *tent status and locations* instead of the three in the given tables.

Table 11: Simple Disclosure Report for the Rendezvous Usecase

	meeting details	mobile location	suggested meetingpoint	tent status and locations A	B	C
PE Android	V	V	-	-	-	-
PULSAR PAL module	V	V	V	-	-	-
RapidGather	-	-	V	-	-	-
Tent Manager A	-	-	-	V	-	-
Tent Manager B	-	-	-	-	V	-
Tent Manager C	-	-	-	-	-	V
Shared over	V	-	-	-	-	-

Table 12: Simple Data Dependency for the Rendezvous Usecase

	meeting details	mobile location	suggested meetingpoint	tent status and locations A	B	C
meeting details	*	-	-	-	-	-
mobile location	-	*	-	-	-	-
suggested meetingpoint	D	D	*	D	D	D
tent status and locations A	-	-	-	*	-	-
tent status and locations B	-	-	-	-	*	-
tent status and locations C	-	-	-	-	-	*

Table 13: Extended Disclosure Report for the Rendezvous Usecase

	meeting details	mobile location	suggested meetingpoint	tent status and locations A	B	C
PE Android	V	O	-	-	-	-
PULSAR PAL module	V	O	V	-, D	-, D	-, D
RapidGather	-, D	-, D	V	-, D	-, D	-, D
Tent Manager A	-	-	-	O	-	-
Tent Manager B	-	-	-	-	O	-
Tent Manager C	-	-	-	-	-	O
Shared over	-	-	-	-	-	-

From these analysis results we can also see that the processes using only generic secure multiparty computation stereotype are very simple by nature. Each party has access to their inputs and only the dedicated parties learn the output. In addition, the output of a secure multiparty computation activity depends on the inputs from all participants in the computation.

A more detailed version of this process can be found in `https://pleak.io/pe-bpmn-editor/viewer/SE2R2wr1S49b_rbaHLhp/` and its analysis results can be seen in Pleak and are discussed in Section 4.5.

4.1.1.3 Crisis Discovery. The crisis discovery scenario uses mobile phone call data to detect if there is a potential crisis occurring. At its simplest, it may just detect if there is a recent emergency call. However, more interesting calling patterns have been discovered for crisis behavior and could be used to get better accuracy. The goal after all is to detect a significant crisis not necessarily single emergency calls. For example, there are known patterns how people calling behavior will change from the regular events when a crisis is at hand. At the moment our analysis only considers the simplest case where an emergency call is detected.

In short, the data from mobile phone is collected periodically and aggregated based on a city block

Table 14: Simple Disclosure Report for Crisis Detection

	privatized city block data	aggregated data 1	aggregated data 2	aggregated privatized data 1	aggregated privatized data 2	call data	call share 1	call share 2	detection output, crisis data	noise scale	time parameters
Command Center	-	-	-	-	-	-	-	-	V	-	-
Mobile device	-	-	-	-	-	V	H	H	-	-	-
PRESNA server	V	-	-	-	-	-	-	-	V	V	V
STEALTH Query Server	V	-	-	H	H	-	-	-	-	V	V
STEALTH server 1	-	H	-	H	-	-	H	-	-	V	V
STEALTH server 2	-	-	H	-	H	-	-	H	-	V	V
Shared over	V	-	-	H	H	-	H	H	V	V	V

of location. The aggregated data with applied differential privacy is used by the PRESNA service to figure out if a crisis has occurred and if so, then where is the crisis location.

The mobile device generates secret shares of its call data and sends these to the PULSAR servers. The PULSAR system aggregates the call data from all users and answers queries from the PRESNA server. In addition, PULSAR system is responsible for applying differential privacy to the aggregation outputs. The PRESNA system runs the crisis detection algorithm on the privatized aggregated data and reports any potential crisis to the command center. The private data in this case is the call data from individual mobile devices. Note that the PULSAR system used here is the same as in the aid tent scenario but it is modeled and analyzed with more details.

The PE-BPMN model for this scenario is available at `https://pleak.io/app/#/view/2eT_Ui356l7w3Qm-a6zE` and on Fig. 49. The following Table 14 and Table 15 show the simple disclosure report and data dependency matrix for this process.

It can be seen that all shares are hiding the input data. However, PE-BPMN editor does not denote that the *privatized city block data* is a result from a differentially private computation and it just gets a *visibility* marker *V* in the disclosure table. From the data dependency matrix we can see that the final output *detection output* depends on all the values computed during the process and all the inputs. Note that the shares are seen in this analysis thanks to considering more details and the technology specific stereotypes instead of the generic secure computation.

4.1.2 Current State and Future Developments of PE-BPMN. Overall we have applied PE-BPMN in various scenarios throughout the Brandeis programme, some further examples of these are covered in Sec. 4.5. In addition, the scenarios considered were the driving force for the design and implementation choices of PE-BPMN. Hence, we have shown that such analysis is feasible in practice and applicable to various scenarios.

Although PE-BPMN at its current state is usable for various cases there are also many details that could be improved or added. Most notably, it might be helpful to add data stereotypes corresponding to all ProtectConfidentiality type task stereotypes that could be used if in fact the input of the process is already protected and for reasonable scope we do not want to add the party doing it to the process at hand. We also started with only allowing one input for PETComputation stereotypes because it matched better with the rest of the analyzers of Pleak. However, in future it would allow for more flexibility of

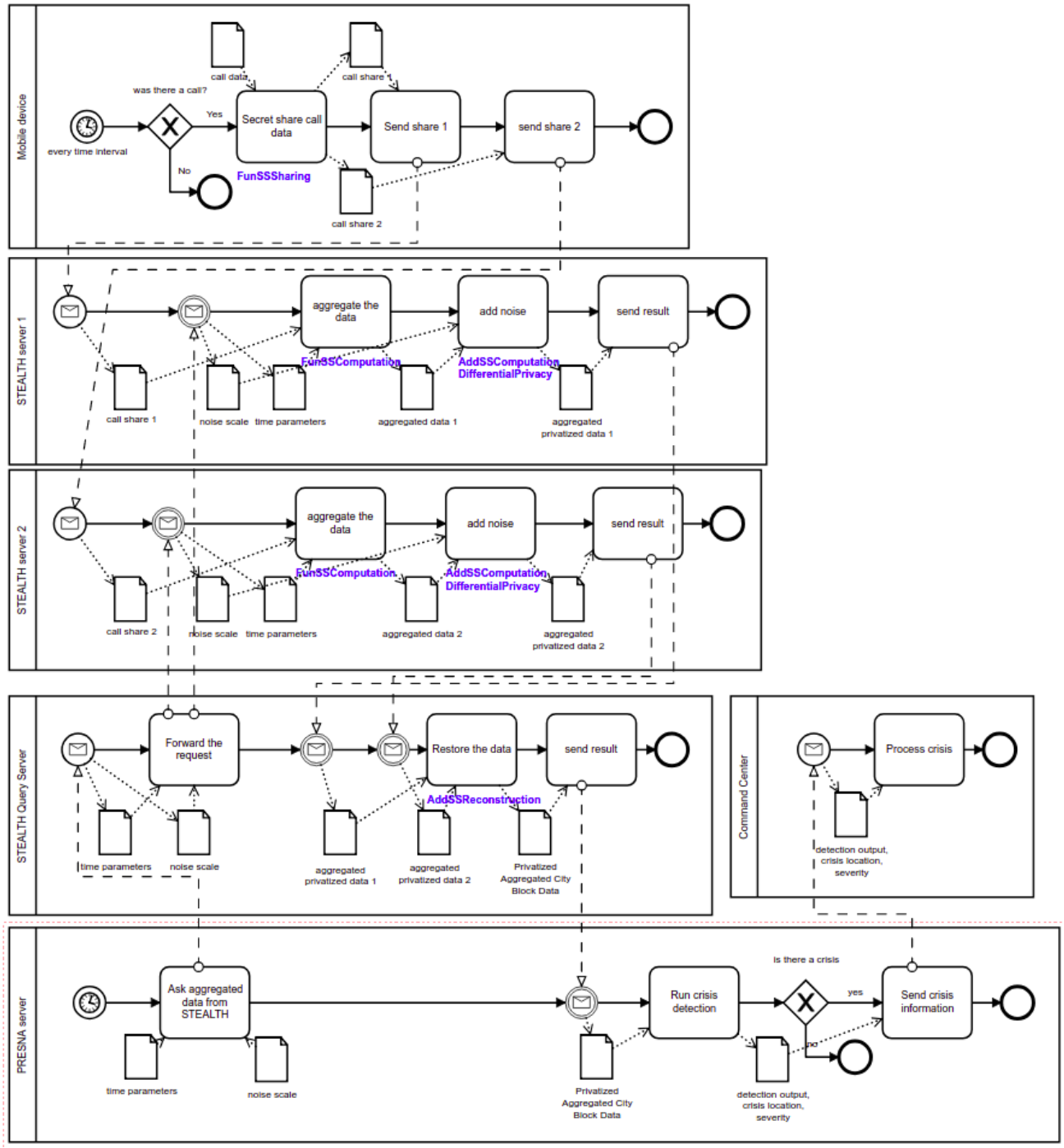


Figure 49: Crisis Detection Process with Different Parties

Table 15: Data Dependency Matrix for Crisis Detection

	privatized city block data	aggregated data 1	aggregated data 2	aggregated privatized data 1	aggregated privatized data 2	call data	call share 1	call share 2	detection output, crisis data	noise scale	time parameters
priv. city block data	*	I	I	D	D	I	I	I	-	I	I
aggregated data 1	-	*	-	-	-	I	D	-	-	I	D
aggregated data 2	-	-	*	-	-	I	-	D	-	I	D
agg. priv. data 1	-	D	-	*	-	I	I	-	-	D	I
agg. priv. data 2	-	-	D	-	*	I	-	I	-	D	I
call data	-	-	-	-	-	*	-	-	-	-	-
call share 1	-	-	-	-	-	D	*	-	-	-	-
call share 2	-	-	-	-	-	D	-	*	-	-	-
detection output	D	I	I	I	I	I	I	I	*	I	I
noise scale	-	-	-	-	-	-	-	-	-	*	-
time parameters	-	-	-	-	-	-	-	-	-	-	*

usage to raise that restriction and instead allow the user to specify types of the computation outputs. Furthermore, it would be beneficial to allow analysing models with collapsed pools. This would give an easy way to analyse your process and the data that is sent out without having to model the receiving side.

4.2 Leaks-When Analysis

The leaks-when analyzer, which we described in Sec. 3.2, has been implemented in OCaml, using the OCamlgraph library[13] for certain transversals of SDGs. The analyzer forms a part of the leaks-when analysis tool, which itself is available as a part of the Pleak tool. The source code of our analyzer back-end is available in GitHub[14]. The implemented and integrated analyses work for collaborative SQL workflows, SQL workflows with policies, and for collaborative business processes specified in BPMN, where the tasks of the process have been annotated with simple expressions defining the values of the *fields* of the datasets that they write into. The values of the fields are also the basis of choosing the flow of the execution at an exclusive gateway.

Both the SQL workflow analysis and the BPMN analysis have been extensively used in NAPLES to study various example processes from other performers in the Brandeis program.

4.2.1 Integration of the Collaborative SQL Workflow Analyzer. The SQL workflow analyzer first combines the SQL statements in the tasks of the business process into SQL workflows, generating several workflows if necessary (see Sec. 3.2.2.3). This component of the analyzer is implemented in Javascript and available in GitHub[15]. This component also implements a Web application that provides a REST endpoint to invoke the Leaks-When analyzer. The Pleak-sql-editor component of Pleak.io uses this REST endpoint to submit SQL workflows for analysis by the Leaks-When analyzer.

[13] `http://ocamlgraph.lri.fr/`

[14] `https://github.com/pleak-tools/pleak-leaks-when-analysis`

[15] `https://github.com/Pleak-tools/Pleak-leaks-when-ast-transformation`

The analyzer accepts a large subset of SQL, with a syntax specific to PostgreSQL. This means that the tool supports a subset of syntax of SQL3, with partial support to some extensions to SQL3 for geographical data, more specifically, to support the computation of distance between two geographical entities based in their coordinates. However, there are also some constraints on the supported syntax, from which the most important ones include:

- The select clause does not support the "*". Instead, the statement must explicitly enumerate the list of attributes that are projected.
- All the tables in the from clause must be associated with aliases. All the attributes in the list of projected attributes must also have an associated alias. Moreover, any reference to attributes must be qualified with the alias of the table, except in the case of the order by clause where attributes must be referred by the alias given in the select clause.

The integration of the front-end and back-end of the analyzer is somewhat peculiar, but the chosen way makes the integration very simple. Namely, the back-end of the analyzer takes no inputs. Instead, the analysis is applied to a workflow defined at compile time in the source file `RAInput.ml` of the analyzer back-end. Hence the analyzed workflow is expressed as an OCaml expression belonging to a variant type, the structure of which is very similar to the structure of our queries and expressions given in Fig. 10 and Fig. 14. In order to execute the analyzer on different workflows, the front-end of the analyzer performs the following steps:

- It translates the SQL workflow into our query language, and replaces the file `RAInput.ml` in the source of our analyzer back-end.
- It recompiles the back-end. We have structured the code of the back-end so, that only the code in the (very short) topmost source file depends on `RAInput.ml`. Hence the object code has to be re-generated only for `RAInput.ml` (and the topmost source file), and not for the parts of the analyzer that generate and simplify the SDG or read the result out of it. This takes very little time.
- It executes our back-end. The outputs of the back-end are placed in certain files, where the name of the folder is given as a command-line parameter.
- It reads the files and displays them to the user.

4.3 Quantitative Analysis

4.3.1 Sensitivity and DP Analysis of Components. We have implemented the analysis described in Sec. 3.3.3. The analysis considers two runs of the analyzed program, with two different versions of the input. For each variable x, let $[\![x]\!]_1$ denote the value of x in the first run and $[\![x]\!]_2$ the value of x in the second run. When considering two runs, information flow properties can be stated as trace properties that can be checked by program analysis. As the language is imperative, the values may change during execution, thus the denoted values are taken at the "current" point, which will be clear from context.

First, the analysis constructs the initial abstract state using the inputs. Then the analysis goes through the program statements in order and updates the abstract state when processing each statement. After that, it uses the abstract state to compute the required distances for output. The abstract state includes the following for each x, where x is a scalar variable or an element of a vector variable:

- (The known bound on) the possible range of x, of the expectation of x, and of the variance of x. For this, the Box domain of the APRON library [70] is used. It stores an n-dimensional (possibly infinite in some dimensions) hypercube where n is 3 times the number of variables (3 dimensions for each variable: the variable itself, its expectation, and its variance). The library is also used to compute the range of a variable assigned to, using the stored ranges of the variables occurring in the assigned expression. The APRON library was originally chosen because it supported a polyhedra domain that we initially used. Unfortunately, the performance was not very good with the polyhedra library:

number of variables	running time (seconds)
100	1.040
200	7.319
400	56.459

We later switched to the Box domain, which had much better performance:

number of variables	running time (seconds)
4000	0.126
10000	0.473
20000	1.009

The Box domain may give worse bounds than the polyhedra domain but in practice, it did not seem to be much worse. Hence we will focus only on the version with the Box domain.

- An upper bound on the *scalar distance* of x, which we define as $|[\![x]\!]_1 - [\![x]\!]_2|$, i.e. how much the value of x may differ in the two runs. Note that if the possible range of x is $[a, b]$ then the scalar distance of x can be at most $b - a$.
- Whether the variable has a Laplace distribution.

For each vector variable v, it includes:

- An upper bound on the *vector distance* of v, which is the number of elements that must be changed in $[\![v]\!]_1$ to transform it into $[\![v]\!]_2$.
- An integer k identifying the *vector distance group* such that for all vectors w in the same group the upper bound d on the vector distance is the same and also the d elements that are possibly changed when transforming $[\![w]\!]_1$ to $[\![w]\!]_2$ can be chosen to be the same for all w.

In addition, the abstract state includes some less important information, e.g. to avoid recomputing and thus improve performance.

4.3.1.1 Implementation. We have implemented the analysis. The analysis considers two runs of the analyzed program, with two different versions of the input. There may be constraints on the allowed range of the input variables and how much the two versions of the input variables may differ. The analysis determines how much (the distributions of) the two versions of some other variables computed in the program differ.

Building and installation. The analyser has been implemented in OCaml. The source of the analyser is available at `https://github.com/pleak-tools/pleak-program-analysis.git`. The analyzer requires the OCaml version of the APRON library. Assuming that OCaml has been installed in the directory `/usr/lib/ocaml` and APRON in the directory `/usr/lib/ocaml/apron`, the analyzer can be built by invoking the following commands at the root directory of the repository:

```
ocamlyacc myparser.mly
ocamllex lexer.mll
ocamlopt -cc "g++" -I /usr/lib/ocaml/apron -o transfer_functions_intervals \
        /usr/lib/ocaml/bigarray.cmxa \
        /usr/lib/ocaml/apron/gmp.cmxa \
        /usr/lib/ocaml/apron/apron.cmxa \
        /usr/lib/ocaml/apron/boxMPQ.cmxa \
        syntax.mli \
        myparser.mli \
        myparser.ml \
        lexer.ml \
        transfer_functions_intervals.ml
```

The source directory of the analyser contains a makefile which may be invoked to execute the building commands. This has been tested on Ubuntu 16.04, where APRON can be installed using the command

```
apt install libapron-ocaml-dev
```

Use. The analyzer is executed by running

```
./transfer_functions_intervals input.txt
```

where the file `input.txt` contains three parts.

The first part (the header) describes constraints on the input variables of the analyzed program. It consists of the following types of declarations, each in a separate line:

- `AddVarInterval` x r_1 r_2
  Specifies that the input variable x (or the elements of it if x is a vector) is in the range $[r_1, r_2]$.
- `AddScalarDist` x d
  Specifies that the input variable x (or the elements of it if x is a vector) differs by at most d in the two versions of the input.
- `AddVecDist` x_1 ... x_k d
  Specifies that the input variables x_i are vectors and when each x_i is considered as the i^{th} column of a matrix X then X differs by at most d rows in the two versions of the input.

The second part contains the analyzed program itself, consisting of statements separated by line breaks. Extra empty lines are allowed. Each statement is of one of the following forms:

- x `:=` e, denoting that the value of the expression e is assigned to the variable x,
- beginning with the line `Repeat` e, followed by a list of statements separated by line breaks, followed by the line `End`, denoting a loop where the list of statements inside the loop is executed n times, where n is the value of the expression e before the loop.

Allowed expressions e are as follows:

$$e ::= x \mid r \mid e_1 \texttt{ + } e_2 \mid e_1 \texttt{ * } e_2 \mid e_1 \texttt{ <= } e_2 \mid e_1 \texttt{ = } e_2 \mid e_1 \texttt{ ? } e_2 \texttt{ : } e_3 \mid \texttt{Sum } e_1 \mid \texttt{Input()} \mid \texttt{Laplace()}$$

denoting variables, constants, addition, multiplication, comparison, equality testing, if-then-else, sum of vector elements, input values, and Laplace noise (with standard deviation 1) generation, respectively.

The third part (the footer) describes the output that we want to get from the analyzer. It consists of the following types of declarations, each in a separate line:

- `PrintVecDist` x_1 ... x_k
  Specifies that the variables x_i are vectors and when each x_i is considered as the i^{th} column of a matrix X then X differs by at most d rows in the two versions of the input, for some d. We want the analyzer to find and output that d.
- `PrintDiffPrivDist` x_1 ... x_k
  Specifies that the variables x_i are scalars and we want the analyzer to find and output an upper bound on the differential-privacy distance between the two versions of the tuple $(x_1, \ldots, x_k)$. It may also be used when some of the x_i are vectors. In this case, the upper bound should hold for any element of x_i used instead of x_i in the tuple, i.e. the tuple still consists of k scalars.

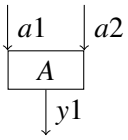
Comments can be included in the input file using the character # that starts a comment lasting until the end of the line.

The directory containing the source code of the analyzer also contains some sample input files `example*.txt`.

4.3.2 Mutual Information in Workflows.

4.3.2.1 Component Types. Here is a (non-exhaustive) list of component types that can be expressed in the system that we described in Sec. 3.3.4. Diagrams of the components are shown on the left and the corresponding declarations read by our analyzer are shown on the right.

Database aggregator




```
comp A a1 a2 -> y1 ;
leak sens 20.0 a1 -> y1 ;
leak sens 50.0 a2 -> y1 ;
```

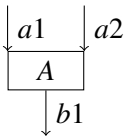
The declarations mean that $(A, a1, y1)$ has 20.0-sensitivity and $(A, a2, y1)$ has 50.0-sensitivity.

The inputs (here $a1$ and $a2$ but in general 1 or more inputs) are database tables and the component aggregates them to a scalar value $y1$. E.g. $y1$ may be the linear correlation coefficient of $a1$ and $a2$. If there is only one input table (e.g. $a1$) then $y1$ may be e.g. the mean, median, or standard deviation of $a1$.

The distance defined on any of its inputs a_i is the number of records by which the two database tables differ. The distance defined on its output $y1$ is the absolute value of the difference between the two scalar values.

For each input a_i, the component has sensitivity $c(a_i, y1)$. E.g. if $y1$ is the mean of a_i and each value in a_i is in the range $[L, R]$ then $c(a_i, y1) = \frac{R-L}{n}$, where n is the number of values (records) in a_i.

Database linker



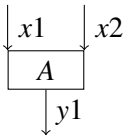

```
comp A a1 a2 -> b1 ;
leak sens 3 a1 -> b1 ;
leak sens 1 a2 -> b1 ;
```

The declarations mean that $(A, a1, b1)$ has 3-sensitivity and $(A, a2, b1)$ has 1-sensitivity.

The database tables $a1$ and $a2$ are linked by a column in each table. Let us call this column the *provenance column* and the possible values in this column the *provenances*. The table $a1$ must have at most one record with each provenance but $a2$ may contain up to r records with each provenance. Then the sensitivities are: $c(a1, b1) = r$ and $c(a2, b1) = 1$. This can be generalized to the case of linking more than 2 tables, of which only one may have non-unique provenances.

The output of a database linker may be used as an input of a database aggregator.

Scalar combiner

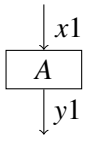



```
comp A x1 x2 -> y1 ;
leak sens 1.0 x1 -> y1 ;
leak sens 1.0 x2 -> y1 ;
```

Here $(A, x1, y1)$ has 1.0-sensitivity and $(A, x2, y1)$ has 1.0-sensitivity. The inputs (2 or more of them, here $x1$ and $x2$) are scalars. They are combined to calculate the output $y1$ (also a scalar).

This can be used to combine outputs of database aggregators. E.g. if $x1$ and $x2$ are the lower and upper quartile, respectively, of a database table then $y1$ may be the difference $x2 - x1$. In this case $c(x1, y1) = c(x2, y1) = 1$.

Laplace randomizer



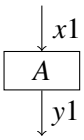

```
comp A x1 -> y1 ;
leak dpr 0.01 x1 -> y1 ;
```

The declarations mean that $(A, x1, y1)$ has 0.01-differential privacy.

The input $x1$ is a scalar value and the output $y1$ is calculated by adding Laplace noise from $Lap(\lambda)$ to $x1$. Here $\frac{1}{\lambda} = \varepsilon(x1, y1) = 0.01$. If $x1$ has sensitivity $\beta_0(x1) = c$ with respect to the global inputs then $\gamma(A, x1, y1) = \frac{c}{\lambda}$.

This can be combined with a database aggregator or scalar combiner to make their result differentially private.

Laplace randomizer without sensitivity

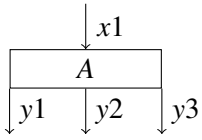



```
comp A x1 -> y1 ;
leak dp 2.0 x1 -> y1 ;
```

Here $(A, x1, y1)$ has sensitivity-less 2.0-differential privacy, with the keyword `leak dp` instead of `leak dpr` indicating that sensitivity is not used.

The input $x1$ is a scalar value and the output $y1$ is calculated by adding Laplace noise from $Lap(\lambda)$ to $x1$. The input does not need to have any sensitivity bound derived from sensitivity declarations. If it does have such a bound, it is ignored. Instead, we assume that $x1$ is in a certain range $[L, R]$ and if it is not there (by some mistake) then it is clipped into that range. Then we add Laplace noise from $Lap(\lambda)$ to $x1$. The result $y1$ is $\frac{R-L}{\lambda}$-differentially private. E.g. we may assume that $x1$ is a result of computing a linear correlation coefficient, being in the range $[-1, 1]$, and take $\lambda = 1$. Then the result is 2-differentially private, i.e. $\gamma(A, x1, y1) = 2$.

Secret sharing




```
comp A x1 -> y1 y2 y3 ;
leak mi 0.0 x1 -> y1 y2 ;
leak mi 0.0 x1 -> y1 y3 ;
leak mi 0.0 x1 -> y2 y3 ;
leak mi 64.0 x1 -> y1 y2 y3 ;
```

The declarations mean that $(A, x1, \{y1, y2\})$ has at most 0.0 bits of mutual information, $(A, x1, \{y1, y3\})$ has at most 0.0 bits of mutual information, $(A, x1, \{y2, y3\})$ has at most 0.0 bits of mutual information, $(A, x1, \{y1, y2, y3\})$ has at most 64.0 bits of mutual information.

Here we secret share $x1$ into three shares $y1, y2, y3$. In the case of additive secret sharing, we would have $y1 \oplus y2 \oplus y3 = x1$, where $\oplus$ is addition modulo 2^k, where k is the bit length of each of the four values.

Here we have information-theoretical bounds on the flows. E.g. $q(x1; y1, y2) = q(x1; y1, y3) = q(x1; y2, y3) = 0$ but $q(x1; y1, y2, y3) = k$.

We can also express other kinds of secret sharing.

4.3.2.2 Implementation. We have implemented (in C++) Alg. 2. The maximum flow from Source to Sink is computed using Edmonds-Karp algorithm. The implementation reads the description of the system, transforms it to a flow network, and finds the maximum flow in this graph. If the system has V components and E wires then the generated directed graph has at most $2V + 2$ nodes and at most $E + V$ edges. Thus the complexity is $O(VE^2)$. It can be improved by using a faster maximum-flow algorithm.

We have also implemented the idea in Sec. 3.3.4.3. We apply the triangle inequality for the inputs and get a bound on $d_M(\mathcal{A}; C)$. We convert it to a bound on $q_M(\mathcal{A}; C)$. We get another bound on $q_M(\mathcal{A}; C)$ using only the known bounds on q_M and monotonicity (triangle inequality cannot be applied here). Either or both of the two bounds may also be infinite (i.e. no bound can be derived). Then we take the minimum of the two bounds.

4.3.2.3 Example. Fig. 50 shows an example of a system with components A, B, C, D and wires $x1, x2, x3, x4, x5, x6, x7$. The corresponding input file describing this system is shown in Fig. 51. This file is read by our implementation.

The file describes the leakages using differential-privacy epsilons, which are shown as 0.2 for each single input and single output of each component. For the component A, we also give the leakage from $\{x1\}$ to $\{x3, x4\}$ because the triangle inequality cannot be used for outputs. The triangle inequality does hold for inputs and it is used to find the leakages involving more than one input of the same component. For example, consider component B. Its leak from $x2$ to $x5$ is 0.2, and from $x3$ to $x5$ is also 0.2. Then its leak from $(x2, x3)$ to $x5$ is $d_B(\{x2, x3\}; \{x5\}) = 0.4$. Then we convert these into upper bounds for the

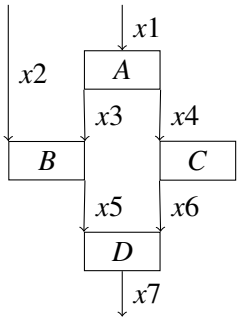


Figure 50: An Example System for Mutual Information Analysis

```
input x1 x2 ;
output x7 ;
comp A x1 -> x3 x4 ;
leak dp 0.2 x1 -> x3 ;
leak dp 0.2 x1 -> x4 ;
leak dp 0.4 x1 -> x3 x4 ;
comp B x2 x3 -> x5 ;
leak dp 0.2 x2 -> x5 ;
leak dp 0.2 x3 -> x5 ;
comp C x4 -> x6 ;
leak dp 0.2 x4 -> x6 ;
comp D x5 x6 -> x7 ;
leak dp 0.2 x5 -> x7 ;
leak dp 0.2 x6 -> x7 ;
check x1 -> x7 ;
check x2 -> x7 ;
check x1 x2 -> x7 ;
```

Figure 51: Input File Describing the System in Fig. 50

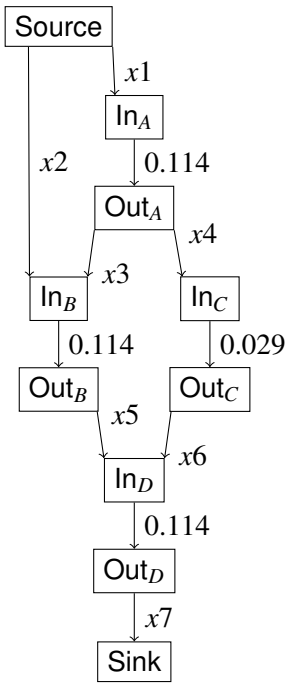


Figure 52: Flow Network From $\{x1, x2\}$ to $\{x7\}$ Corresponding to the System in Fig. 50

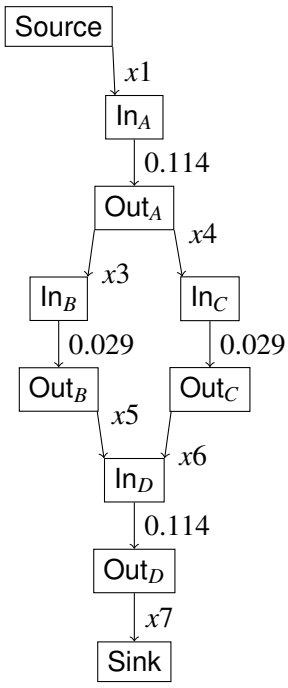


Figure 53: Flow Network From $\{x1\}$ to $\{x7\}$ Corresponding to the System in Fig. 50

mutual-information-based leakages:

$$\begin{aligned} q_B(\{x2, x3\}; \{x5\}) &\leq 0.114 \\ q_B(\{x2\}; \{x5\}) &\leq 0.029 \\ q_B(\{x3\}; \{x5\}) &\leq 0.029 \end{aligned}$$

As we see, the triangle inequality does not hold for q_B.

Then a flow network for a subset of the global inputs and outputs is generated for the system. The result for the input subset $\{x1, x2\}$ and the output subset $\{x7\}$ is shown in Fig. 52. The wires with finite capacity have their capacity shown next to them, instead of their name. The direction of the edges is downwards. We find the maximum flow from Source to Sink, which is 0.114.

Considering the input subset $\{x1\}$ and the output subset $\{x7\}$, we get the flow network in Fig. 53. The capacity of the edge from In_B to Out_B is now 0.029 instead of 0.114, Reducing the maximum flow from Source to Sink to 0.058.

We also find the maximum flow from the input subset $\{x2\}$ to the output subset $\{x7\}$, getting 0.029. Thus the triangle inequality also does not hold for the global system, as $0.029 + 0.058 < 0.114$.

4.3.3 Derivative Sensitivity w.r.t. Components. The analysis of Sec. 3.3.8 has been implemented in Haskell and is available on GitHub[16]. As an input, it takes an SQL query, a database schema, and a description of the norm w.r.t. which we want to achieve differential privacy. We assume that each table contains a row *ID* of unique keys. For each table X, we expect a table named *X_sensRows* that contains the same column *ID* of keys, and another column *sensitive* of boolean values that tell for each row whether it is sensitive or not.

The analyzer computes *another query* (as a string) that describes the way in which derivative sensitivity should be computed. This new query represents the function $c(x)$ such that the additive noise would be $\frac{c(x)}{b} \cdot \eta$ for $\eta \leftarrow \mathit{GenCauchy}(\gamma)$, according to Theorem 3.62. In our analyzer, $\gamma = 4$ is fixed, and $b = \epsilon/(\gamma + 1) - \beta$, where ϵ is the desired differential privacy level, and β the smoothness parameter, which is provided as an additional input. The resulting query is fed to a database engine to evaluate the sensitivity on particular data.

4.3.3.1 Evaluation. We performed evaluation on 4 x Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz laptop, Ubuntu 16.04.4 LTS, using PostgreSQL 9.5.14.

We have taken the queries of TPC-H set [71] for benchmarking. Most of these queries contain GROUP BY constructions with too many possible groups. We have simplified these queries, adding a filtering that chooses *one* particular group.

Another challenge comes from the filters. If some filter is "public" (i.e. does not depend on sensitive data), it is easier to apply it beforehand, so that the remaining table with "private" filters (that do depend on sensitive data and hence cannot be applied directly) would be as small as possible. While it is easy to do with a pure AND combination of filters, in practice public and private filters can be mixed, e.g. related by OR. We had to manually rewrite the filters in such a way that public filters would be easily extractable as separate members of an AND combination.

We generated TPC-H data with scale factors (SF) 0.1, 0.5, 1.0, denoting how much data is generated for the sample database. For 1.0, the size of the largest table is ca 6 million rows. The table schema, together with numbers of rows for different tables, is given in Sec. 4.3.3.2. To define the database metric, we have considered integer, decimal, and date columns as sensitive, assigning to them different weights, described more precisely in Sec. 4.3.3.3. All rows are considered sensitive. Row norms have been combined using ℓ_1-norm, which ensures differential privacy w.r.t. unit change in sensitive attribute of one row.

We adjusted (as described above) the queries $Q1$ (splitting a single query with 5 aggregations to 5 separate queries), $Q2$ (splitting it to 2 queries with MIN and MAX respectively), $Q3$-$Q11$, $Q12$ (splitting

[16] `https://github.com/pleak-tools/pleak-sql-analysis/tree/master/banach`

2 aggregations to 2 queries), $Q16$, $Q17$, $Q19$ of the TPC-H dataset to our analyzer. The queries that have been eventually fed to the analyzer are listed in Sec. 4.3.3.4. We treat date as an integer, i.e. the number of days passed from the date 1980-01-01. In Sec. 4.3.3.5, we present more evaluation results, where we treat date as a floating point number, so that sigmoids can be used for filtering.

We fix $\epsilon = 1$. For derivative sensitivity experiments, we take sigmoid precision $\alpha = 5$ and smoothness $\beta = 0.1$. This choice gives $b = 0.1$, and the additive noise with 78% probability is below $10 \cdot c(x)$, where the value 78% comes from analyzing distribution *GenCauchy*(4) (as discussed in Sec. 3.3.8.5). Too large value of β makes b (and hence the noise) larger, and too small β makes the sensitivity larger, so β is a parameter that can in general be optimized.

Time. The time benchmarks are given in Table 16. Let x be the database instance. For each scale factor SF, we report the execution time t_i of the *initial query* $q_i(x)$, time t_m of the *modified query* $q_m(x)$ (i.e. in which filtering is replaced with continuous approximation), and time t_s of the *sensitivity-computing query* $q_s(x)$. The time spent to generate the queries q_m and q_s is negligible (below $20ms$), and it does not depend on the database size, so we do not report it. We also do not report the execution time of sampling the noise, as it does not depend on the database size either.

The total time overhead of computing noisy output based on derivative sensitivity is $t_m + t_s$: since the sensitivity has been computed for q_m, the noise should also be added to $q_m(x)$, and not to $q_i(x)$. We estimate the total time overhead for global sensitivity as t_i, as it is sufficient to execute $q_i(x)$, and the computation of global sensitivity does not depend on the database size.

We see that in general t_m and t_s are larger than t_i. This overhead comes from filtering. While in q_i the database engine may immediately drop all rows that do not satisfy the filter, in q_m and q_s we need to compute the approximated output and the sensitivity of *each row*. In overall, the time overhead of q_m and q_s compared to q_i (and hence of derivative sensitivity compared to global sensitivity) depends on the ratio of "number of rows before filtering" and "number of rows after filtering".

Precision. The precision benchmarks are given in Table 17. For each scale factor SF, we report the output $q_i(x)$ of the initial query, and $q_s(x)$ of the sensitivity-computing query. We report the output $q_m(x)$ of the modified query only if it is different from $q_i(x)$. The relative error has been computed as $\frac{|(q_m(x)\pm\xi)-q_i(x)|}{q_i(x)} \cdot 100$, where $\xi = \frac{c(x)}{b} = \frac{q_s(x)}{\epsilon/(\gamma+1)-\beta} = 10 \cdot q_s(x)$. The additive noise stays below ξ with probability 78% (as discussed above), so the relative error stays below reported value also with probability 78%.

The last two columns of Table 17 demonstrate the global sensitivity of queries, which is the same for all SF values, as it does not depend on data. The left column shows global sensitivity w.r.t. the same metric as the derivative sensitivity (we call it *non-standard*), and the right column w.r.t the row difference metric (we call it *standard*). We compare these with derivative sensitivity.

Global sensitivity w.r.t. non-standard metric. In the first case, we compute the global sensitivity w.r.t. the same metric as the derivative sensitivity. Even using the same metric, we cannot compare global (GS) and derivative sensitivity (DS) directly without taking into account particular noise generating mechanisms. However, in our results we have either GS=∞ or GS=DS. If GS=∞, then the noise would be ∞ as well for any noise generating mechanism. If GS=DS, then we expect the noise of GS to be lower, as e.g. employing the same Cauchy mechanism that we use for derivative sensitivity with $\beta \approx 0$ gives 10 times less noise than with $\beta = 0.1$ for the same sensitivity. In our benchmarks, DS gives advantage over GS in the following main cases.

1. When a sensitive attribute x_1 is multiplied by another attribute x_2, and there are no bounds on x_2, we get GS=∞, as $|(x_1 \pm 1) \cdot x_2 - x1 \cdot x2| = |x_2|$.
2. The norms of rows are combined into a table norm using ℓ_1-norm. Hence, $d(t, t') = 1$ covers not only the case where the norm of one row changes by 1, but also the case where *each* row changes a little. In an extreme case, all rows of t are already very close to the filtering bound, and the filtering function returns ≈ 1 for all rows in t, and ≈ 0 for all rows in t'. This makes no difference

Table 16: Time Benchmarks (ms) for the Initial Query (t_i), Modified Query (t_m), and the Sensitivity Query (t_s). K Denotes $\cdot 10^3$

	SF = 0.1			SF = 0.5			SF = 1.0		
	t_i	t_m	t_s	t_i	t_m	t_s	t_i	t_m	t_s
$Q1\_1$	152.8	534.59	763.19	731.11	3.17K	4.08K	1.5K	5.6K	8.01K
$Q1\_2$	151.8	559.58	1.04K	1.47K	4.02K	5.86K	1.62K	6.59K	11.92K
$Q1\_3$	168.08	590.1	2.05K	862.07	3.24K	10.37K	1.75K	5.87K	19.66K
$Q1\_4$	184.24	574.28	2.2K	888.35	2.98K	10.08K	1.69K	5.96K	20.45K
$Q1\_5$	149.96	527.38	520.85	744.5	2.69K	2.86K	1.48K	5.46K	5.86K
$Q2\_1$	19.68	45.3	144.78	134.21	294.14	1.04K	289.53	563.79	2.25K
$Q2\_2$	29.04	49.37	165.62	158.94	273.06	1.28K	288.18	632.38	2.49K
$Q3$	111.92	117.41	391.47	544.22	623.87	2.19K	349.06	521.31	1.2K
$Q4$	131.52	379.05	778.47	799.9	2.63K	5.16K	1.56K	5.05K	10.69K
$Q5$	6.66K	204.08	2.18K	696.38	685.59	3.61K	1.51K	2.2K	9.71K
$Q6$	118.31	3.12K	13.21K	687.4	16.09K	67.29K	1.26K	31.73K	123.64K
$Q7$	238.74	137.21	713.28	1.19K	861.55	3.9K	2.42K	1.67K	8.34K
$Q8$	308.08	117.53	782.37	1.3K	1.73K	5.89K	4.08K	1.45K	8.38K
$Q9$	133.34	128.58	3.82K	1.79K	728.07	4.21K	1.59K	1.42K	9.17K
$Q10$	131.97	137.03	483.38	882.12	719.65	2.46K	202.05	1.48K	4.88K
$Q11$	10.74	10.16	42.12	62.0	62.02	254.81	126.47	128.67	529.29
$Q12\_1$	215.13	736.64	1.27K	879.2	3.65K	7.5K	1.95K	7.34K	14.04K
$Q12\_2$	148.5	473.72	877.42	846.66	3.26K	6.19K	2.44K	4.8K	10.84K
$Q16$	22.14	174.35	303.68	127.95	711.93	1.63K	264.52	1.66K	3.66K
$Q17$	111.7	88.31	276.69	486.16	455.85	1.38K	938.62	1.12K	2.96K
$Q19$	139.16	296.41	1.42K	737.53	1.47K	6.67K	1.39K	2.86K	13.56K

for a COUNT query (as in $Q1\_5$), as the sum of all these changes is still 1, but we get GS=∞ for the query $Q1\_1$, which has the same form as $Q1\_5$, except that it is a SUM query.

Global sensitivity w.r.t. standard metric. In the second case, we compute global sensitivity w.r.t. row difference metric. That is, $d(x, x') = 1$ iff there is exactly one sensitive table in databases x and x' such that the respective instances t and t' of that table differ in one row. To make the comparison more fair, we consider an input table sensitive iff the query uses at least one of its attributes that were considered sensitive by the ℓ_p-metric. For SUM, MIN, MAX queries, the effect of adding/removing a row is unbounded, and global sensitivity is ∞, as it covers the worst case. For COUNT queries, we may lose advantage as well if we consider a JOIN of tables, where adding/removing a row in an input table may result in adding/removing an unbounded number of rows in the cross product of input tables, as it happens in $Q4$. Row difference metric gives smaller sensitivity in the COUNT-queries $Q12\_1$, $Q12\_2$, $Q16$. In general, if we filter by a sensitive attribute over a single input table, then row difference metric contributes 1 to the COUNT, while defining the distance as ℓ_1-norm of rows allows to split the unit change among several rows, which may result in higher sensitivity.

4.3.3.2 Database Schema. The TPC-H testset [71] puts forth the following database schema, as given below. The tables are (randomly) filled with a number of rows, generated by a program that accompanies the schema. The number of rows depends on the *scaling factor SF*. The tables, and the numbers of rows in them are given in Fig. 54.

4.3.3.3 Sensitive Components. In all tables except **Lineitem**, we consider the change that is the scaled sum of changes in all sensitive attributes. All attributes that are not a part of the norm are considered insensitive. We assumed that textual fields as well as the keys (ordinal data) are not sensitive. The columns of type `date` (e.g. *o_orderdate*) have been converted to a floating-point number, which is the number of months passed from the date 1980-01-01.

Part: $SF \cdot 200,000$ rows.

column	type
P_PARTKEY	identifier
P_NAME	text
P_MFGR	text
P_BRAND	text
P_TYPE	text
P_SIZE	integer
P_CONTAINER	text
P_RETAILPRICE	decimal
P_COMMENT	text

Customer: $SF \cdot 150,000$ rows.

column	type
C_CUSTKEY	identifier
C_NAME	text
C_ADDRESS	text
C_NATIONKEY	identifier
C_PHONE	text
C_ACCTBAL	decimal
C_MKTSEGMENT	text
C_COMMENT	text

Partsupp: $SF \cdot 800,000$ rows.

column	type
PS_PARTKEY	identifier
PS_SUPPKEY	identifier
PS_AVAILQTY	integer
PS_SUPPLYCOST	decimal
PS_COMMENT	text

Supplier: $SF \cdot 10,000$ rows.

column	type
S_SUPPKEY	identifier
S_NAME	text
S_ADDRESS	text
S_NATIONKEY	identifier
S_PHONE	text
S_ACCTBAL	decimal
S_COMMENT	text

Orders: $SF \cdot 1,500,000$ rows

column	type
O_ORDERKEY	identifier
O_CUSTKEY	identifier
O_ORDERSTATUS	text
O_TOTALPRICE	decimal
O_ORDERDATE	date
O_ORDERPRIORITY	text
O_CLERK	text
O_SHIPPRIORITY	integer
O_COMMENT	text

Lineitem: $SF \cdot 6,000,000$ rows

column	type
L_ORDERKEY	identifier
L_PARTKEY	identifier
L_SUPPKEY	identifier
L_LINENUMBER	integer
L_QUANTITY	decimal
L_EXTENDEDPRICE	decimal
L_DISCOUNT	decimal
L_TAX	decimal
L_RETURNFLAG	text
L_LINESTATUS	text
L_SHIPDATE	date
L_COMMITDATE	date
L_RECEIPTDATE	date
L_SHIPINSTRUCT	text
L_SHIPMODE	text
L_COMMENT	text

Nation: 25 rows

column	type
N_NATIONKEY	identifier
N_NAME	text
N_REGIONKEY	identifier
N_COMMENT	text

Region: 5 rows

column	type
R_REGIONKEY	identifier
R_NAME	text
R_COMMENT	text

Figure 54: Schema and Size of TPC-H Database

Table 17: Precision Benchmarks for $\epsilon = 1, \beta = 0.1$, Sigmoid $\alpha = 5$, where $q_i(x)$ is the Initial Query Result, $q_m(x)$ the Modified Query Result (if Different from $q_i(x)$), $q_s(x)$ is the Sensitivity Query Result, and %noise $= \frac{|(q_m(x) \pm 10 \cdot q_s(x)) - q_i(x)|}{q_i(x)} \cdot 100$. The Last Two Columns Show Global Sensitivity w.r.t. the Same Non-Standard ℓ_p-Induced Metric as Derivative Sensitivity (non-std.) and the Standard "Row Difference" Metric (std.). K Denotes $\cdot 10^3$, M Denotes $\cdot 10^6$, and G Denotes $\cdot 10^9$

	SF = 0.10			SF = 0.50			SF = 1.00			global sens.	
	$q_i(x)$ ($q_m(x)$)	$q_s(x)$	%noise	$q_i(x)$ ($q_m(x)$)	$q_s(x)$	%noise	$q_i(x)$ ($q_m(x)$)	$q_s(x)$	%noise	non-std.	std.
$Q1\_1$	3.79M	50.0	0.01	18.87M	50.0	0.0026	37.72M	50.0	0.0013	∞	∞
$Q1\_2$	5.34G	95.89K	0.02	27.35G	99.65K	0.0036	56.57G	104.9K	0.0019	∞	∞
$Q1\_3$	5.07G	107.36K	0.02	25.98G	111.18K	0.0043	53.74G	117.34K	0.0022	∞	∞
$Q1\_4$	5.27G	114.87K	0.02	27.02G	119.06K	0.0044	55.89G	124.38K	0.0022	∞	∞
$Q1\_5$	148.3K	1.0	0.0067	739.56K	1.0	0.0014	1.48M	1.0	0.0007	1	1
$Q2\_1$	1.07	100.0	93.46K	1.0	100.0	100.0K	1.0	100.0	100.0K	100	∞
$Q2\_2$	999.98	100.0	100.0	1.0K	100.0	100.0	1.0K	100.0	100.0	100	∞
$Q3$	3.62K	41.28K	11.4K	3.21K	41.1K	12.8K	0.0	0.0	0.0	∞	∞
$Q4$	2.92K	7.0	2.4	14.17K	7.0	0.49	28.07K	7.0	0.25	7	∞
$Q5$	5.37M	260.44K	48.53	25.23M	359.6K	14.25	47.6M	484.12K	10.17	∞	∞
$Q6$	17.45M (17.13K)	125.0K	7.14	88.13M (86.86M)	127.0K	1.44	181.93M (179.15)	130.0K	0.71	∞	∞
$Q7$	22.07M	106.13K	4.81	95.63M	111.24K	1.16	212.11M	115.33K	0.54	∞	∞
$Q8$	470.8K	145.15K	308.31	2.74M	172.5K	63.04	3.29M	178.96K	54.4	∞	∞
$Q9$	30.32M	40.0K	1.32	137.73M	49.2K	0.36	283.82M	49.2K	0.17	∞	∞
$Q10$	100.31K	357.71K	3.57K	149.6K	398.13K	2.66K	0.0	312.54K	∞	∞	∞
$Q11$	1.63G	199.98K	0.12	7.73G	199.98K	0.03	15.18G	199.98K	0.01	∞	∞
$Q12\_1$	3.12K	3.0	0.96	15.4K	3.0	0.19	30.83K	3.0	0.1	3	1
$Q12\_2$	1.29K	3.0	2.33	6.2K	3.0	0.48	12.37K	3.0	0.24	3	1
$Q16$	9.95K	4.0	0.4	49.35K	4.0	0.08	98.97K	4.0	0.04	4	1
$Q17$	31.54K (31.17K)	16.8K	533	256.24K (250.83K)	17.8K	69.3	531.93K (520.87K)	18.0K	33.9	∞	∞
$Q19$	155.25K	651.72K	4.2K	1.1M	813.52K	738.04	1.73M	827.69K	479.67	∞	∞

- **Part:** $\|p\_size, 0.01 \cdot p\_retailprice\|_1$. The values of *p_retailprice* are measured in hundreds, so we consider larger changes (i.e. make such change causing a change of 1 in the output correspond to unit sensitivity).
- **Partsupp:** $\|ps\_availqty, 0.01 \cdot ps\_supplycost\|_1$.
- **Orders:** $\|30 \cdot o\_orderdate, 0.01 \cdot o\_totalprice\|_1$.
- **Customer:** $\|0.01 \cdot c\_acctbal\|_1$.
- **Supplier:** $\|0.01 \cdot s\_acctbal\|_1$.
- **Nation:** no sensitive columns.
- **Region:** no sensitive columns.

In table **Lineitem**, several different norms would make sense and it is up to the data owner to choose the "right" one. We could again add up the sensitive attributes of a row, after suitably scaling them. But we could also think that the three different dates would probably move rather synchronously, and it is the maximum change among them that really matters. Hence we performed the tests with the row norm

$$\|l\_quantity, 0.0001 \cdot l\_extendedprice, 50 \cdot l\_discount, 30 \cdot \|l\_shipdate, l\_commitdate, l\_receiptdate\|_\infty\|_1 \ .$$

Here the values of *l_discount* are very small (all around 0.1), so we aim to protect the change in 0.02 units. On the other hand, *l_extendedprice* can be tens of thousands, and we want to capture larger changes for it. The dates are measured in *months*, so we capture a change of one day. which we treat as $1/30$ of a month. Alternatively, we may use the number of *days*, and get the same result removing the scaling by 30.0.

4.3.3.4 Benchmarked Queries. We list the rewritten queries of TPC-H dataset that were used in benchmarking, to give an impression of what we actually feed to the analyzer. The constant 0.142857 in Q_17 comes from translating an AVG query to a SUM query, and $7 = 1/0.142857$ is the number of rows to sum, which is public, as the filter does not use sensitive attributes.

```
--Q1_1
SELECT SUM(lineitem.l_quantity)
FROM lineitem
WHERE
  lineitem.l_shipdate <= 230.3 - 30
  AND lineitem.l_returnflag = 'R'
  AND lineitem.l_linestatus = 'F'
;
--Q1_2
SELECT SUM(lineitem.l_extendedprice)
FROM lineitem
WHERE
  lineitem.l_shipdate <= 230.3 - 30
  AND lineitem.l_returnflag = 'R'
  AND lineitem.l_linestatus = 'F'
;
--Q1_3
SELECT SUM(lineitem.l_extendedprice
           *(1-lineitem.l_discount))
FROM lineitem
WHERE
  lineitem.l_shipdate <= 230.3 - 30
  AND lineitem.l_returnflag = 'R'
  AND lineitem.l_linestatus = 'F'
;
--Q1_4
SELECT SUM(lineitem.l_extendedprice
           *(1-lineitem.l_discount)
           *(1+lineitem.l_tax))
FROM lineitem
WHERE
  lineitem.l_shipdate <= 230.3 - 30
  AND lineitem.l_returnflag = 'R'
  AND lineitem.l_linestatus = 'F'
;
--Q1_5
SELECT COUNT(*)
FROM lineitem
WHERE
  lineitem.l_shipdate <= 230.3 - 30
  AND lineitem.l_returnflag = 'R'
  AND lineitem.l_linestatus = 'F'
;
--Q2_1
SELECT MIN(partsupp.ps_supplycost)
FROM partsupp, supplier,
     nation, region, part
WHERE
  part.p_partkey = partsupp.ps_partkey
  AND supplier.s_suppkey = partsupp.ps_suppkey
  AND supplier.s_nationkey = nation.n_nationkey
  AND nation.n_regionkey = region.r_regionkey
  AND region.r_name = 'ASIA'
;
--Q2_2
SELECT MAX(partsupp.ps_supplycost)
FROM partsupp, supplier,
     nation, region, part
WHERE
  part.p_partkey = partsupp.ps_partkey
  AND supplier.s_suppkey = partsupp.ps_suppkey
  AND supplier.s_nationkey = nation.n_nationkey
  AND nation.n_regionkey = region.r_regionkey
  AND region.r_name = 'ASIA'
;
--Q3
SELECT SUM(lineitem.l_extendedprice
```

```
            *(1-lineitem.l_discount))
FROM customer, orders, lineitem
WHERE
  customer.c_mktsegment = 'BUILDING'
  AND customer.c_custkey = orders.o_custkey
  AND lineitem.l_orderkey = orders.o_orderkey
  AND orders.o_orderdate < 190
  AND lineitem.l_shipdate > 190
  AND lineitem.l_orderkey = '162'
  AND orders.o_shippriority = '0'
;
--Q4
SELECT COUNT(*)
FROM orders, lineitem
WHERE
  orders.o_orderdate >= 180
  AND orders.o_orderdate < 180 + 3
  AND lineitem.l_orderkey = orders.o_orderkey
  AND lineitem.l_commitdate < lineitem.l_receiptdate
  AND orders.o_orderpriority = '1-URGENT'
;
--Q5
SELECT SUM(lineitem.l_extendedprice
            *(1-lineitem.l_discount))
FROM customer, orders,
     lineitem, supplier,
     nation, region
WHERE
  customer.c_custkey = orders.o_custkey
  AND lineitem.l_orderkey = orders.o_orderkey
  AND lineitem.l_suppkey = supplier.s_suppkey
  AND customer.c_nationkey = supplier.s_nationkey
  AND supplier.s_nationkey = nation.n_nationkey
  AND nation.n_regionkey = region.r_regionkey
  AND region.r_name = 'ASIA'
  AND orders.o_orderdate >= 213.3
  AND orders.o_orderdate < 213.3 + 12
  AND nation.n_name = 'JAPAN'
;
--Q6
SELECT SUM(lineitem.l_extendedprice
            *lineitem.l_discount)
FROM lineitem
WHERE
  lineitem.l_shipdate >= 170.5
  AND lineitem.l_shipdate < 170.5 + 12
  AND lineitem.l_discount BETWEEN 0.09 - 0.01
                          AND 0.09 + 0.01
  AND lineitem.l_quantity < 24
;
--Q7
SELECT SUM(lineitem.l_extendedprice
            *(1 - lineitem.l_discount))
FROM supplier, lineitem, orders, customer,
     nation as n1,
     nation as n2
WHERE
  supplier.s_suppkey = lineitem.l_suppkey
  AND orders.o_orderkey = lineitem.l_orderkey
  AND customer.c_custkey = orders.o_custkey
  AND supplier.s_nationkey = n1.n_nationkey
  AND customer.c_nationkey = n2.n_nationkey
  AND (
    (n1.n_name = 'JAPAN' and n2.n_name = 'INDONESIA')
    OR
    (n1.n_name = 'INDONESIA' and n2.n_name = 'JAPAN')
  )
  AND lineitem.l_shipdate between 182.6 and 207
;
--Q8
SELECT SUM(lineitem.l_extendedprice
        *(1 - lineitem.l_discount))
FROM
```

```
  part, supplier, lineitem,
  orders, customer,
  nation AS n1, nation AS n2, region
WHERE
  part.p_partkey = lineitem.l_partkey
  AND supplier.s_suppkey = lineitem.l_suppkey
  AND lineitem.l_orderkey = orders.o_orderkey
  AND orders.o_custkey = customer.c_custkey
  AND customer.c_nationkey = n1.n_nationkey
  AND n1.n_regionkey = region.r_regionkey
  AND region.r_name = 'ASIA'
  AND supplier.s_nationkey = n2.n_nationkey
  AND orders.o_orderdate >= 5478
  AND orders.o_orderdate <= 6210
  AND part.p_type = 'MEDIUM␣BRUSHED␣COPPER'
  AND n2.n_name = 'JAPAN'
;
--Q9
SELECT SUM(lineitem.l_extendedprice
           *(1-lineitem.l_discount)
       - partsupp.ps_supplycost*lineitem.l_quantity)
FROM
    part,     supplier,
    lineitem, partsupp,
    orders,   nation
WHERE
  supplier.s_suppkey = lineitem.l_suppkey
  AND partsupp.ps_suppkey = lineitem.l_suppkey
  AND partsupp.ps_partkey = lineitem.l_partkey
  AND part.p_partkey = lineitem.l_partkey
  AND orders.o_orderkey = lineitem.l_orderkey
  AND supplier.s_nationkey = nation.n_nationkey
  AND part.p_name LIKE '%violet%'
  AND nation.n_name = 'UNITED␣KINGDOM'
;
--Q10
SELECT SUM(lineitem.l_extendedprice
           *(1 - lineitem.l_discount))
FROM
  customer, orders,
  lineitem, nation
WHERE
  customer.c_custkey = orders.o_custkey
  AND lineitem.l_orderkey = orders.o_orderkey
  AND orders.o_orderdate >= 183.3
  AND orders.o_orderdate <  183.3 + 3
  AND lineitem.l_returnflag = 'R'
  AND customer.c_nationkey = nation.n_nationkey
  AND customer.c_custkey = '64'
  AND nation.n_name = 'CANADA'
;
--Q11
SELECT SUM(partsupp.ps_supplycost
           * partsupp.ps_availqty * 0.2)
FROM partsupp, supplier, nation
WHERE
  partsupp.ps_suppkey = supplier.s_suppkey
  AND supplier.s_nationkey = nation.n_nationkey
  AND nation.n_name = 'JAPAN'
;
--Q12_1
SELECT COUNT(*)
FROM orders, lineitem
WHERE
  orders.o_orderkey = lineitem.l_orderkey
  AND (orders.o_orderpriority <> '1-URGENT'
       OR orders.o_orderpriority <> '2-HIGH')
  AND lineitem.l_shipmode in ('TRUCK', 'SHIP')
  AND lineitem.l_commitdate < lineitem.l_receiptdate
  AND lineitem.l_shipdate < lineitem.l_commitdate
  AND lineitem.l_receiptdate >= 183.3
  AND lineitem.l_receiptdate < 183.3 + 12
;
```

```
--Q12_2
SELECT COUNT(*)
FROM orders, lineitem
WHERE
  orders.o_orderkey = lineitem.l_orderkey
  AND (orders.o_orderpriority = '1-URGENT'
       OR orders.o_orderpriority = '2-HIGH')
  AND lineitem.l_shipmode in ('TRUCK', 'SHIP')
  AND lineitem.l_commitdate < lineitem.l_receiptdate
  AND lineitem.l_shipdate < lineitem.l_commitdate
  AND lineitem.l_receiptdate >= 183.3
  AND lineitem.l_receiptdate < 183.3 + 12
;
--Q16
SELECT COUNT(partsupp.ps_suppkey)
FROM partsupp, part, supplier
WHERE
  part.p_partkey = partsupp.ps_partkey
  AND partsupp.ps_suppkey = supplier.s_suppkey
  AND part.p_brand <> 'Brand#34'
  AND NOT (part.p_type LIKE '%COPPER%')
  AND part.p_size in (5, 10, 15, 20, 25, 30, 35, 40)
  AND NOT (supplier.s_comment LIKE
                           '%Customer%Complaints%')
  AND part.p_brand = 'Brand#14'
  AND part.p_type  = 'LARGE␣ANODIZED␣TIN'
;
--Q17
SELECT SUM(lineitem.l_extendedprice * 0.142857)
FROM lineitem, part
WHERE
  part.p_partkey = lineitem.l_partkey
  AND part.p_brand = 'Brand#34'
  AND part.p_container = 'JUMBO␣PKG'
  AND lineitem.l_quantity < 0.2 * 32
;
--Q19
SELECT SUM(lineitem.l_extendedprice
           *(1-lineitem.l_discount))
FROM lineitem, part
WHERE
  part.p_partkey = lineitem.l_partkey
  AND lineitem.l_shipmode IN ('AIR', 'AIR␣REG')
  AND lineitem.l_shipinstruct = 'DELIVER␣IN␣PERSON'
  AND part.p_size >= 1
  AND
  ((
    part.p_brand = 'Brand#34'
    AND part.p_container IN ('SM␣CASE', 'SM␣BOX',
                             'SM␣PACK', 'SM␣PKG')
    AND lineitem.l_quantity >= 35
    AND lineitem.l_quantity <= 35 + 10
    AND part.p_size <= 5
    )
    OR
    (
    part.p_brand = 'Brand#22'
    AND part.p_container IN ('MED␣BAG', 'MED␣BOX',
                             'MED␣PKG', 'MED␣PACK')
    AND lineitem.l_quantity >= 12
    AND lineitem.l_quantity <= 12 + 10
    AND part.p_size  <= 10
    )
    OR
    (
    part.p_brand = 'Brand#14'
    AND part.p_container IN ('LG␣CASE', 'LG␣BOX',
                             'LG␣PACK', 'LG␣PKG')
    AND lineitem.l_quantity >= 90
    AND lineitem.l_quantity <= 90 + 10
    AND part.p_size  <= 15
));
```

Table 18: Time Benchmarks (ms) for the Initial Query (t_i), Modified Query (t_m), and the Sensitivity Query (t_s). K Denotes $\cdot 10^3$, and M Denotes $\cdot 10^6$

	SF = 0.1			SF = 0.5			SF = 1.0		
	t_i	t_m	t_s	t_i	t_m	t_s	t_i	t_m	t_s
$Q1\_1$	144.36	11.43K	1.74K	761.36	157.38K	9.49K	1.47K	535.97K	18.51K
$Q1\_2$	141.57	11.43K	1.74K	742.35	163.8K	9.57K	1.46K	518.45K	19.08K
$Q1\_3$	154.66	11.75K	1.89K	886.02	157.73K	9.93K	1.67K	538.28K	21.61K
$Q1\_4$	165.84	11.64K	2.81K	851.84	154.48K	15.4K	1.97K	558.75K	29.17K
$Q1\_5$	149.43	6.6K	1.06K	769.5	65.65K	6.16K	1.43K	189.94K	11.0K
$Q2\_1$	19.08	36.87	139.14	141.1	268.5	1.28K	269.94	524.61	2.24K
$Q2\_2$	19.11	42.29	135.36	146.93	278.97	1.07K	265.5	724.23	2.07K
$Q3$	96.29	110.48	376.65	567.6	645.44	2.15K	288.72	451.16	1.03K
$Q4$	127.72	59.0K	2.37K	653.65	1.8M	12.67K	–	–	–
$Q5$	6.25K	242.1	2.13K	715.05	877.75	3.37K	1.32K	2.49K	7.96K
$Q6$	115.36	284.1K	9.37K	586.04	6.34M	46.5K	–	–	–
$Q7$	218.87	329.15	623.63	1.16K	2.93K	3.74K	2.45K	9.05K	7.13K
$Q8$	238.62	113.11	637.68	1.15K	719.8	3.9K	3.43K	1.22K	6.54K
$Q9$	127.56	121.25	3.23K	746.92	719.1	4.22K	1.29K	1.3K	7.83K
$Q10$	129.26	134.37	445.86	676.09	790.71	2.4K	205.47	1.45K	4.92K
$Q11$	10.49	9.87	38.92	62.34	61.11	244.71	125.13	126.73	524.31
$Q12\_1$	157.39	110.13K	4.62K	849.04	2.49M	26.3K	–	–	–
$Q12\_2$	146.71	44.21K	2.12K	727.03	961.28K	10.6K	–	–	–
$Q16$	21.45	130.13	216.23	155.18	735.92	1.65K	237.25	1.42K	3.38K
$Q17$	86.52	83.86	269.3	481.85	463.56	1.43K	872.76	861.25	2.55K
$Q19$	130.1	264.35	1.21K	718.78	1.48K	7.74K	1.3K	2.72K	12.73K

4.3.3.5 Integer vs Float Type Filtering. Since the *date* datatype of SQL is measured within day precision, it makes sense to treat is as an integer. However, we could as well represent it as a floating-point number. This allows us to use sigmoids, as shown in Table 9. For sigmoids, we have to choose precision in such a way that the noise would be smaller. Since precision itself cannot depend on the data, we have empirically evaluated appropriate precision level on an independently generated TPC-H instance with scale factor SF=0.05. As described in Sec. 3.3.8.5, the precision has to be increased proportionally with $\sqrt{n}$, where n is the number of analyzed rows. Hence, the sigmoid precisions for the cases of SF 0.1, 0.5, 1.0 had to be multiplied with $\sqrt{2}$, $\sqrt{10}$ and $\sqrt{20}$ respectively.

While Table 16 and Table 17 use integer approximation for date filtering, the tables Table 18 and Table 19 show the results for sigmoid approach. The results have been computed for different β and α values, where $\beta \approx 0$ means that the sensitivity could be computed for an arbitrarily small β, and the third column shows the base α that has been computed for SF=0.05. The computing time for a modified query is much higher for floating points, since the SQL engine now needs to compute exponentiation for each row and each private filter, so for the most complicated queries we present the results up to SF=0.5. We see that, except the queries $Q12\_1$ and $Q12\_2$, the error gets smaller compared to integer datatype approach. The problem of $Q12\_1$ and $Q12\_2$ seems to be that the sigmoid precision that we found for SF=0.05 is not the best for SF=0.1, which indeed may happen as the final result depends not only on the number of rows, but also on the actual data, so even though the sensitivity of these queries is smaller, they suffer from precision error. The disadvantages of sigmoid approach are that it takes more time to execute the modified query, and that exponentiations tend to cause overflow errors in PostgreSQL engine when the exponents get large. The time overheads are more significant for the cases with many private filters, where sigmoid error gets larger due to multiplication, so it seems more reasonable to use integer datatype there.

4.3.3.6 Discussion. Let us summarize the limits and advantages of the framework proposed in this work. We compare ℓ_p-metric vs row distance metric, and local sensitivity vs global sensitivity. In the following, we mark with + the clear advantages, and with − some caveats.

Table 19: Precision Benchmarks for $\epsilon = 1$, where $q_i(x)$ is the Initial Query Result, $q_m(x)$ the Modified Query Result (if Different from $q_i(x)$), $q_s(x)$ is the Sensitivity Query Result, and %noise = $\frac{|(q_m(x) \pm 10 \cdot q_s(x)) - q_i(x)|}{q_i(x)} \cdot 100$. K Denotes $\cdot 10^3$, M Denotes $\cdot 10^6$, and G Denotes $\cdot 10^9$

			SF = 0.1			SF = 0.5			SF = 1.0		
	β	sigmoid prec. α	$q_i(x)$ ($q_m(x)$)	$q_s(x)$	%noise	$q_i(x)$ ($q_m(x)$)	$q_s(x)$	%noise	$q_i(x)$ ($q_m(x)$)	$q_s(x)$	%noise
Q1_1	≈ 0	0.1	3.79M	1.8	0.0002	18.87M	4.03	0.0001	37.72M	5.7	7.6e-05
Q1_2	≈ 0	0.1	5.34G	10.0K	0.0009	27.35G	10.0K	0.0002	56.57G	11.18K	9.9e-05
Q1_3	≈ 0	0.1	5.07G	19.0K	0.0019	25.98G	19.0K	0.0004	53.74G	21.24K	0.0002
Q1_4	0.1	0.1	5.27G	12.11K	0.0023	27.02G	12.11K	0.0004	55.89G	13.91K	0.0002
Q1_5	≈ 0	0.05	148.3K	0.02	6e-05	739.56K	0.04	2.7e-05	1.48M	0.06	1.9e-05
Q2_1	0.1	0.1	1.07	100.0	93.46K	1.0	100.0	100.0K	1.0	100.0	100.0K
Q2_2	0.1	0.1	999.98	100.0	100.0	1.0K	100.0	100.0	1.0K	100.0	100.0
Q3	≈ 0	0.01	3.62K (2.1K)	19.0K	2.58K	3.21K (2.17K)	19.0K	2.93K	0.0	0.0	0.0
Q4	≈ 0	0.5	2.92K (2.96K)	1.24	1.73	14.17K	3.32	0.76	–	–	–
Q5	0.1	0.01	5.37M (4.82M)	11.21K	8.1	25.23M (24.74M)	11.21K	1.5	47.6M (46.88M)	11.21K	1.27
Q6	≈ 0	0.4	17.45M (17.91M)	105.0K	5.67	88.13M (89.71M)	105.0K	1.81	–	–	–
Q7	≈ 0	0.1	22.07M (21.99M)	19.0K	0.06	95.63M (95.44M)	19.0K	0.1	212.11M (212.02M)	21.24K	0.0078
Q8	0.1	0.1	470.8K (470.86K)	11.21K	23.83	2.74M (2.75M)	13.64K	5.31	3.29M (3.3M)	20.01K	6.44 6.44
Q9	0.1	0.1	30.32M	40.0K	1.32	137.73M	49.2K	0.36	283.82M	49.2K	0.17
Q10	0.1	0.1	100.31K (100.06K)	12.65K	125.84	149.6K (143.52K)	31.48K	206.34	0.0	34.94K	∞
Q11	0.1	0.1	1.63G	199.98K	0.12	7.73G	199.98K	0.03	15.18G	199.98K	0.01
Q12_1	≈ 0	0.5	3.12K (3.38K)	0.53	8.43	15.4K (16.52K)	1.19	7.32	–	–	–
Q12_2	≈ 0	0.5	1.29K (1.4K)	0.53	8.88	6.2K (6.67K)	1.19	7.61	–	–	–
Q16	0.1	0.1	9.95K	4.0	0.4	49.35K	4.0	0.08	98.97K	4.0	0.04
Q17	≈ 0	0.5	31.54K (40.57K)	2.53K	68.66	256.24K (253.19K)	5.65K	9.83	531.93K (519.68K)	7.99K	5.21
Q19	0.1	0.1	155.25K	651.72K	4.2K	1.1M	813.52K	738.04	1.73M	827.69K	479.67

Applicability.

+ Metrics induced by ℓ_p-norms allow to state different privacy goals, and can be useful in cases where the standard row distance metric is not applicable.
– Computation of derivative sensitivity requires a particular data instance. This is similar to local sensitivity. Since execution of the sensitivity-computing query can be deferred, and the data will anyway be needed at the point where a noisy output is released, we do not treat it as an applicability issue.
– Derivative sensitivity is limited to continuous functions. This is not a problem as far as there exist efficiently computable approximations. We can still cover a wide range of SQL queries.

Complexity.

+ In the first phase of the analysis, the initial query it transformed to sensitivity-computing query. The execution time of this transformation is negligible and does not depend on the data.
– In the second phase of the analysis, when the output is ready to be released, we need to execute the sensitivity-computing query to estimate amount of noise. Compared to the initial query, additional time overhead comes from filtering, as we cannot ignore the rows that have been discarded by the filter.

Amount of noise

+ Changing a numeric attribute of a row in general has smaller effect on the query result than adding/removing an entire row.

+ As global sensitivity always covers the worst-case data instance, it is in general larger than local and derivative sensitivity.
– Compared to global sensitivity w.r.t. standard metric, there are more parameters to be tuned in order to optimize the amount of noise, such as smoothness and sigmoid precision.
– While ℓ_p-norms allow to define a variety of metrics over databases, they are not a superset of standard metrics, and for some privacy goals we can get less noise using standard metric.

Adding noise *before* filtering is the path towards solving the issue of complexity and noise overhead that comes from filtering over sensitive attributes.

4.3.4 Derivative Sensitivity w.r.t. Row Multiplicities. The analysis of Sec. 3.3.7 has been implemented in Haskell and is available on GitHub[17]. As an input, it takes a SQL query and a database schema (given as files), and the concrete database (in PostgreSQL) for which to compute local sensitivity.

It first generates another SQL query that for each use of a table, computes the patterns (corresponding to the potential added/removed rows) together with the derivatives w.r.t. each pattern. It also computes some intermediate data needed for smoothing the derivatives (the smoothing is completed in Haskell). This query is executed in the database engine. If some table is used more than once then the unification (described in Sec. 3.3.7.3) of the patterns of the different uses will be done in Haskell because it would be difficult to implement in SQL. We chose to compute the initial patterns for each table use in SQL to leverage the optimizations and support for more SQL expressions built into the PostgreSQL database engine.

For each input table, the analyzer computes the noise level to add to the query result to obtain differential privacy w.r.t. adding/removing a row in that table. Then the maximum of those levels is taken to obtain DP w.r.t. adding/removing a row in any input table. The noise level can be computed for a generalized Cauchy distribution (obtaining ϵ-DP) or Laplace distribution (obtaining (ϵ, δ)-DP).

We have run our analyzer on TPC-H [71] datasets and queries (counting the number of rows they return), where the queries have been in the form (31), using a computer with four-core Intel i7-8550U CPU with 1.8 GHz clock speed, 16 GB RAM, running Ubuntu 18.04.4 and PostgreSQL 10.12. We generated database instances with different scale factors (SF; defined by TPC-H), ran our analyser, invoked the SQL queries it generated, and compiled the results. The database schema of TPC-H dataset is the same as in Section 4.3.3.2. Differently from Sec. 4.3.3, we did not have to adjust any queries manually.

Many queries of TPC-H aggregate using the SUM operation. We treat them as COUNT-s; as we do not know a priori the upper limit of the values being summed, we just assume one (1000). Some queries of TPC-H involve a single table (Q1, Q6), and hence always have their sensitivity equal to 1. On the other hand, the smooth sensitivity of the queries joining $n \geq 3$ tables is huge; the hugeness is caused by the smoothing, because the quantity added to local sensitivity in order to smoothen it is basically the product of the numbers of rows in the largest $(n - 2)$ tables.

Queries involving exactly two tables. The queries Q12, Q14, Q17, and Q19 join exactly two tables. For these queries, the smooth sensitivity does not strongly depend on the scale factor. For the smoothness level $\beta = 0.1$, we have found that an upper bound to the smooth sensitivity of both parts of Q12 is ≈ 5, and to the second part of Q14 is $\approx 4.5 \cdot 10^5$. The "parts" of the query are the different aggregations that it makes. The difference in the level of smooth sensitivity comes from the fact that Q12 is a counting query, while Q14 is a summing query.

The query Q17 splits into three parts, once the nested query has been transformed out. The last of them involves three tables, and hence has large sensitivity. An upper bound to the smooth sensitivity of the first part (a summing query) is ≈ 1500, and of the second part (a counting query) ≈ 56.

The query Q19 (a sum query) also consists of three parts, once the disjunctive WHERE-clause has been split. Each of them has the sensitivity of ca. $4.2 \cdot 10^5$.

[17] `https://github.com/pleak-tools/pleak-sql-analysis/tree/master/localsensitivity-cabal`

Table 20: Running Time of the Analyzer (in Seconds)

SF	Q5	Q7a	Q8a	Q10	Q11	Q12a	Q12b	Q14b	Q17a	Q17b	Q17c	Q19a	Q19b	Q19c
1.0					16							115	117	117
0.1					0.6	405	911	44	68	67	131	2.0	2.0	2.0
0.01		100			0.1	5.3	9.5	1.3	2.0	1.9				
0.005			217											
0.002	13.9		10.6											
0.001	0.3		1.4	45										

Efficiency of the analyzer. The running time of the analyzer is obviously strongly dependent on the size of the database. We have run a selection of the TPC-H queries (those that our analyzer can support) on various scale factors and measured their running times. The results are given in Table 20. In our experience, most of the time is spent on smoothening the sensitivity, executing the SQL queries that compute the counts for the patterns.

Discussion. We have presented a method to compute smooth sensitivity of counting queries specified in SQL, and evaluated the applicability of the method. We find that the currently the applicability is limited. We believe that it could be increased by providing some information about the uniqueness of certain values in the database (e.g. which attributes are the key of a table). Also, the applicability could perhaps be increased by doing the noise addition elsewhere, not necessarily at the end of executing the query.

Nevertheless, our method is a novel approach to securing database queries, and it may work well in conjunction of other techniques for adding the noise, or estimating the magnitude of the query result on similar databases. Our tool uses results of Sec. 3.3.9 to combine derivative sensitivities w.r.t. row multiplicities and w.r.t. components, thus covering a larger variety of SQL queries that can be analyzed.

4.3.5 Guessing Advantage. Guessing advantage analysis of Sec. 3.3.10 is integrated directly into the derivative sensitivity analysis tool described in Sec. 4.3.3. This component converts the desired upper bound ϵ' on guessing advantage to an appropriate ϵ as well as the database norm for differential privacy. The derivative sensitivity tool then estimates the noise distribution.

While the particular analysis result depends on the query and the data, there are some general results that we depict in Figure 55. We are using the method of estimating ϵ from Sec. 3.3.10.1. The first plot computes the formula

$$\epsilon = \frac{-\ln\left(\frac{p}{1-p} \cdot (\frac{1}{x} - 1)\right)}{R} \ ,$$

where x is the desired posterior guessing probability (the guessing advantage plus the prior), p is the prior guessing probability (different priors correspond to different collored lines) and $R = 1$ (we explain this choice below). We get $\epsilon = 0$ for $x < p$ since regardless of how much noise we add, the posterior guessing probability cannot be smaller than the prior. The worst-case prior is $p = 1$ since no noise will help if the attacker already knows everything in advance.

The second plot computes the formula

$$\epsilon = \frac{-\ln\left(\frac{p}{1-p} \cdot (\frac{1}{x+p} - 1)\right)}{R}$$

for $p \leq 0.5$, and

$$\epsilon = \frac{-\ln\left(\frac{1-p}{p} \cdot (\frac{1}{x+1-p} - 1)\right)}{R}$$

for $p > 0.5$, where x is now the desired upper bound on the guessing advantage. We get $\epsilon = \infty$ for $x > 1 - p$, which basically means that the attacker is allowed to guess everything and we do not need

noise for protection. The worst-case prior is defined as $p = \frac{1-x}{2}$ for the particular guessing advantage x, as explained in Sec. 3.3.10.1. Again, we take $R = 1$.

Let us now explain how the results can be interpreted.

Discrete norm: the value $R = 1$ is suitable for the setting where the attacker is guessing whether a record belongs to the database or not. It is also suitable for guessing a particular discrete attribute. In this case, $p = \Pr[\mathbf{X} = x]$ is the probability of an exact guess where x is the actual input.

Continuous norm: taking $R = 1$ means that we have scaled the underlying Banach space by $\frac{1}{\sup_{x,x' \in X} d(x,x')}$ (assuming that the supremum exists). In this case, $p = \Pr[\mathbf{X} \in X']$ is the probability of a "sufficiently close guess", where the "closeness" in terms of distance depends on the distribution of inputs. E.g. if the distribution is uniform, then $X' = \{x' \mid d(x, x') \leq p\}$.

We could actually take any R. We see that, taking $R = 1000$ would also make the ϵ values 1000 times smaller. This, however, does not mean that the corresponding DP mechanism would have to introduce more noise. In particular, the amount of noise is proportional to $\frac{1}{\epsilon}$ as well as sensitivity of the query, and the sensitivity in turn depends on the underlying metric. Scaling the metric by 1000 scales both the ϵ and the sensitivity by 1000, so the noise level remains the same.

4.3.6 Model-Checking Sensitivity of SQL Queries.

4.3.6.1 Queries. The analyzer described in Sec. 3.3.13 has been implemented in Haskell. The source of the analyzer is available at `https://github.com/pleak-tools/pleak-sql-analysis.git`. We have tested our analyzer with all of the 22 queries from TPC-H [71]. The front-end of our analyzer does not support full SQL, hence we have made some non-essential modifications to the queries and schemas from this dataset. These modifications, described in Sec. 4.3.6.2, do not change the logic of computations and tests of the queries, nor their sensitivity.

Several of TPC-H queries contain subqueries. It is possible to analyze them using workflows, i.e. the subquery and the main query are analyzed separately and combined using a workflow. TPC-H queries without subqueries are Q1, Q3, Q5, Q6, Q10, and Q19.

Now we describe the transformations used to transform the original query to a workflow:

(i) If the subquery is in the `FROM`-part then the result table of the subquery is an input table of the main query. An example of this type of subquery is Q7.

(ii) If the subquery is in the `WHERE`-part and returns a single value then the result table of the subquery must represent the functional dependence of the result of the subquery on the keys of the free table variables in the subquery. The tables containing the columns corresponding to the keys are added to the `SELECT`- and `GROUP BY`- part of the subquery. The result table of the subquery is joined to the input tables of the main query using the columns corresponding to the free variables of the subquery. An example of this type of subquery is Q2.

(iii) If the subquery is used in the `WHERE`-part as an argument of the `EXISTS`-operator (testing whether the subquery returns at least 1 row) then we can instead take the `COUNT` of the result of the original subquery as the single-value result of the modified subquery and check whether this value is greater than 0. Then we can reduce this query to the case handled in Transformation ii. An example of this type of subquery is Q4.

(iv) If the subquery is used in the `WHERE`-part as an argument of the `IN`-operator (testing whether a value belongs to the set returned by the subquery) then the result table of the subquery is an input table of the main query and testing whether a value belongs to this intermediate table is replaced with testing whether the value is equal to the only attribute of the intermediate table (now joined to the rest of the input tables). An example is Q18.

(v) If the subquery is used in the `WHERE`-part as an argument of the `NOT IN`-operator (testing whether a value does not belong to the set returned by the subquery) and the value is an attribute with a foreign key constraint referencing the attribute returned by the subquery then we can negate the

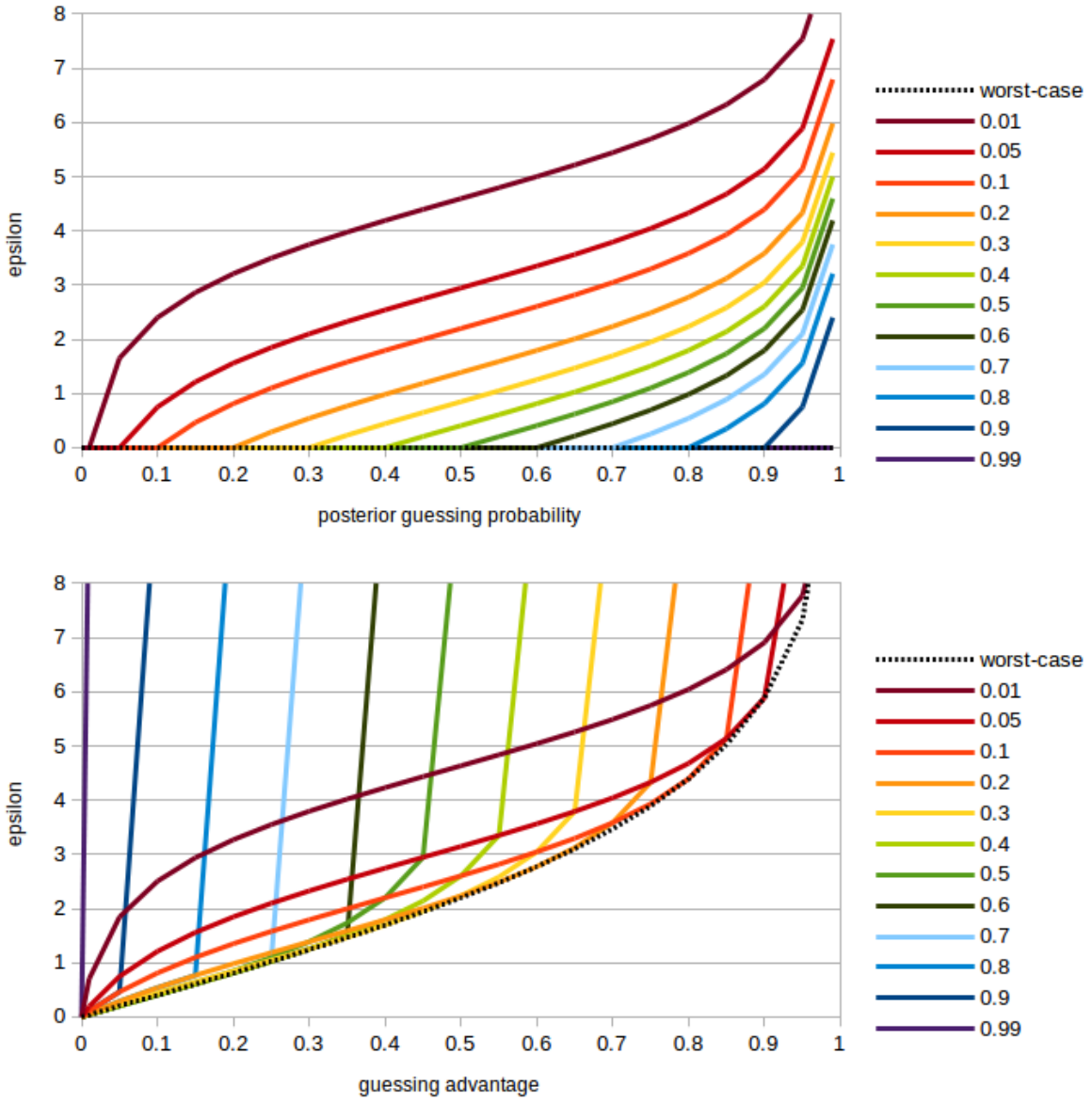


Figure 55: Relation of Posterior Guessing Probability (Top) and Guessing Advantage (Bottom) vs ϵ for Different Priors (Represented by Different Colorful Lines)

WHERE-part in the subquery and replace NOT IN with IN, then transform it as in Transformation iv. An example is Q16.

(vi) If a GROUP BY query has a HAVING-clause (not supported by our analyzer) then this can be replaced by the query with the HAVING-clause removed and an extra query that filters the result of the GROUP BY query using a WHERE-clause. An example is Q18. A simpler alternative that always produces at least as good upper bounds, is to analyze the query with the HAVING-clause removed, and take its sensitivity as an upper bound on the sensitivity of the original query. This is correct because the filtering performed by the HAVING-clause has sensitivity at most 1.

(vii) CASE WHEN expression is not supported by our analyzer but in the TPC-H queries it occurs only in arguments of aggregating functions (in queries Q8, Q12, and Q14). As seen from Sec. 3.3.13.8, the analysis of GROUP BY queries does not use the aggregating functions (P_A) in the analysis. For non-GROUP BY aggregating queries (Q14), the analysis can use P_A but we have not implemented this and our actual implementation uses 1 as the upper bound on the sensitivity. Thus in both cases we can replace the CASE WHEN expression by an arbitrary supported expression (e.g. constant 0) and this does not change the sensitivity determined by the analyzer.

(viii) Query Q15 uses a temporary VIEW. This can be replaced by an ordinary query in the workflow.

(ix) Query Q13 uses a LEFT OUTER JOIN (not supported by our analyzer). We were not able to transform it into an equivalent supported query. Thus we instead replaced the LEFT OUTER JOIN by an ordinary join and proved manually that both the original subquery and the changed one have global sensitivity 1 w.r.t. each of the tables customer and orders. The proof is omitted due to lack of space.

The transformations were performed manually, they are not implemented in the analyzer.

Table 21 lists for each TPC-H query the numbers of the transformations that we applied to it, the number of queries in the workflow obtained after the transformations, and the total running time of the analyzer on the queries in the workflow. The total running time was 12.206 s.

We tested with our analyzer all of the queries in the workflows obtained from the 22 TPC-H queries. All of these had sensitivities 1 or ∞ w.r.t. each table, except Q12, which has sensitivity 2 (note that this query returns at most 2 rows). To get a finite sensitivity greater than 1 for a query with unbounded result size, we modified Q10 slightly, adding o_orderdate to the SELECT- and GROUP BY-parts and reducing the interval of o_orderdate in the WHERE-part from 3 months to N days. The resulting query has sensitivity N and its semantics is not too artificial (it considers a period of N days instead of 3 months and finds the lost revenue for each day separately instead of for the whole period; this can be reasonable e.g. in a scenario where the customers are supermarkets making orders from a big warehouse several times per day). The analyzer terminates in reasonable time (less than 5 seconds) when $N \leq 7$.

Thus all 22 TPC-H queries can be analyzed after the transformations described above and the modifications described in Sec. 4.3.6.2.

4.3.6.2 Modifications to Our Test Queries. The TPC-H queries use a number of different data types and operations. For our benchmarking, we have reduced the variety, rewriting the schemas and queries as follows:

- For all numeric attributes, we have used the types INT8 and FLOAT8 for integers and decimals, respectively.
- Expressions of the form x IN $(x_1, \ldots, x_n)$ are replaced with $(x = x_1$ OR ... OR $x = x_n)$.
- Some more modifications whose descriptions we omit due to lack of space.

The TPC-H queries also contain ORDER BY parts. While the ordering of the result can leak information, this leakage is not considered by our analyzer, which considers the result table as a set or multiset of rows where order is not important (we may shuffle the rows of the result table uniformly randomly). Thus we omit the ORDER BY parts from the queries.

Table 21: Query Evaluation Summary and Running Times

Query	transformations	workflow size	runtime (s)
Q1	—	1	0.170
Q2	ii	2	1.600
Q3	—	1	0.288
Q4	iii	2	0.247
Q5	—	1	0.403
Q6	—	1	0.098
Q7	i	2	1.119
Q8	i, vii	2	1.632
Q9	i	2	1.331
Q10	—	1	0.357
Q11	ii, vi	3	0.656
Q12	vii	1	0.179
Q13	i, ix	2	0.240
Q14	vii	1	0.131
Q15	ii, viii	3	0.311
Q16	v	2	0.355
Q17	ii	2	0.239
Q18	iv, vi	3	0.461
Q19	—	1	0.153
Q20	ii, iv, iv	4	0.593
Q21	iii, iii	3	1.118
Q22	i, ii, iii	4	0.507

4.3.6.3 Performance. The analyzer can determine whether a query has a sensitivity at most, or greater than a certain integer value. To determine the exact value of the sensitivity, it does a linear search trying $0, 1, 2, \ldots$ until it finds a value N such that the sensitivity is at most N. If for all values $n \in \{0, \ldots, N-1\}$, the sensitivity was greater than n, we can conclude that the sensitivity is equal to N.

To find N, we use linear search (instead of binary) because the running time increases very fast as N increases. For the modified TPC-H query Q10, with different values for the parameter that determines sensitivity, the running time for checking that the sensitivity is at most N, where N is the actual sensitivity, was

N	8	7	6	5	4
running time (s)	140.5	2.693	0.377	0.260	0.129

The running time for checking that the sensitivity is more than $(N-1)$, was between 0.1 and 0.2 seconds in all cases. Thus most of the running time is spent on the final check and the linear search does not hurt performance much. The timings (both here and in Sec. 4.3.6.1) were taken on a laptop with an Intel Core i5-2520M CPU running at 2.5 GHz, and with 2GB of RAM. Almost all of this time was spent by the Z3 solver looking for a model for the formula Φ. The time for generating the formula was negligible.

The highest finite sensitivity of a query for which the analyzer terminated in less than 5 minutes, was 9. For most queries with sensitivity 9, the analyzer did not terminate in 5 minutes. Thus the highest finite sensitivity of a simple query that can be practically determined is about 8 or 9. If the solver does not terminate in 5 minutes for a query, we abort it and return ∞ as an upper bound for the sensitivity.

For composite queries (using set operations), the determined sensitivity can be higher, as long as each simple subquery has sensitivity at most 8 or 9. For queries using a table more than once, we can also obtain a higher upper bound on sensitivity if we use the method described in Sec. 3.3.13.6.

We terminate the linear search when we have checked all the values up to 10 and obtained that the sensitivity is greater than 10. Because we have not seen a simple query with a finite sensitivity greater than 10, we take ∞ as the sensitivity in this case. This makes determining infinite sensitivity reasonably fast.

Determining uniqueness constraints. Finding if $\{a_{i_1}, \ldots, a_{i_k}\}$ is a uniqueness constraint on the result table of a query, uses $N = 2$ in the formula for the SMT solver, and is therefore similar to determining whether the sensitivity of a query is at most 1 or more than 1. Thus it should be fast. However, the total number of possible uniqueness constraints to check, is 2^m where m is the number of columns in the result table, i.e. it is exponentially large in the number of columns. To keep the running time reasonable even if m is more than 3 or 4, the current implementation only checks single-column uniqueness constraints, i.e. those of the form $\{a_{i_1}\}$.

4.3.6.4 Using Tables More Than Once. In Sections 3.3.13.4 and 3.3.13.6, we describe two ways of computing the sensitivity of a query w.r.t. a table used more than once. We have implemented both of these. To see the difference, consider the example

```
SELECT t1.y, t2.y FROM t as t1, t as t2
WHERE t2.x >= t1.x AND t2.x < t1.x + 5
```

with the schema

```
CREATE TABLE t (x int4 PRIMARY KEY,
                y int4 NOT NULL);
```

The query in this example has sensitivity 9. The algorithm in Sec. 3.3.13.4 returns the correct sensitivity in 35 seconds. The algorithm in Sec. 3.3.13.6 takes only 0.3 seconds to run but returns 10 as the sensitivity. Thus the former is more accurate but much slower, the latter is less accurate but much faster. For some other queries, the former does not terminate at all in reasonable time but the latter quickly computes an upper bound on (but not the exact value of) the sensitivity.

4.4 Pleak Tool

Pleak tool contains most of the analysis outlined in this report. They are accessible though different editors and components of the tool as also illustrated by Pleak architecture in Fig. 47. This section summarizes the expected usage and restrictions proposed by the current implementation of Pleak. The source code of Pleak is available in `https://github.com/pleak-tools` and a live version is running in `https://pleak.io`.

4.4.1 Using Pleak. Pleak is an open source tool for analyzing the privacy in business processes and pointing out possible leakages. Using Pleak starts with modeling the business processes in Business Process Model and Notation (BPMN) language. BPMN models should be human readable and should focus on concrete subprocess at a time so that their correctness can be verified by the subject matter experts. However, analyzing privacy leakages from the process requires a bigger picture of how the different processes interact with each other. Pleak bridges the gap between human readability and extensive details needed for formal analysis with its composition tool. This tool enables the analyst to easily combine the human readable models into an analyzable model. It is especially valuable when different part of the process can be verified by different experts who can focus on their part of the model and we can bring everything together for the analysis.

The first layer of Pleak's analysis - the boolean level of who sees what in the process - can already be carried out on the pure BPMN model. However, very often this means that we see leakages that are really avoided by some technical means, e.g. secure network channels. Hence, Pleak uses privacy-enhanced BPMN (PE-BPMN) that allows to specify different data protection means that are applied,

such as encrypting data for storage or secure network channels. In PE-BPMN some tasks are specified as tasks of some privacy preserving technology - e.g. a task for encryption and a task for decryption. PE-BPMN model is a good basis for a more thorough look at the privacy of the system. Our automated analysis outputs a table that summarizes which data is seen by which party. For everything a party sees we also analyze which inputs of the process might affect the view of the party. Essentially, this layer of analysis is for understanding that some things can not leak to some parties. However, for everything that is in any way accessible to a party we should dig deeper.

We have devised two flavors of leaks-when analysis to get a deeper look on the conditions of potential leakages. Using them also requires the analyst to specify more about the actual contents of the data processing. Either the tasks are specified using SQL scripts or our own very simple pseudocode. For SQL queries the semantics is very clear and therefore the analysis of the process combines and simplifies the queries to present the user with graphs of the computation. For each field in the data object of interest we can see exactly which fields of which input data are used to derive it and how (the leaks part of the analysis). In addition we can also see all the conditions that must hold in order for the specific value to appear in the output (aka. the when part of the analysis). As a way of simplification you can think of the output as being the select and where part of a SQL query if the process was represented as one query. On the other hand our own pseudocode has some dedicated functions but offers more freedom for the user to specify different flavors of processes. In this analysis the focus is on the branching in the process and on operations that perform any sort of filtering (as specified by the user). Hence, the output shows for each output data field the predicates that need to hold for some input data to affect this output and which filters have been applied to the inputs before they may flow to the output. Hence, leaks-when analysis offers the user more insight to the structure of the computations than a simple BPMN model could. However, it is still a further question if the chosen filters are sufficient to limit the potential leakage.

Pleak's third layer of analysis works on quantifying the leakage either in terms of sensitivity of the process or the adversary's advantage of guessing some private input when observing the outputs of the process. Since we are looking to quantify the variation of the output then these analysis are usually applicable when we have a workflow that outputs some numeric value. In addition, most analysis in this category benefit from having access to the actual data in addition to the process and Pleak allows users to enhance input data objects with the underlying data. Sensitivity tells us how much the output of the process may change given some small change in the input - either a change in the value or adding or removing rows in input tables. On the other hand, guessing advantage analysis allows the user to specify questions about how much does the adversary gain from seeing some of the process outputs when the goal of the adversary is to guess something about the inputs. Furthermore, we can propose restrictions about the guesses, e.g. to specify that there is some reasonable interval where the guess has to be anyway.

We have collected information regarding Pleak's usage to our wiki in `pleak.io/wiki`. In addition, we have recorded a series of screen cast videos that showcase and explain the usage of different Pleak components.

- Pleak interface, editors and analysers – `https://www.youtube.com/watch?v=jr39R3NgSPg`
- Introduction to PE-BPMN – `https://www.youtube.com/watch?v=ihUdxKRM3fU`
- PE-BPMN streotype usage – `https://www.youtube.com/watch?v=rNwbOJSl7X8`
- Composition modeler – `https://www.youtube.com/watch?v=9nzZh8PimHA`
- Summary of Pleak's components and usage ideas – `https://www.youtube.com/watch?v=mk8d4CuobWs`

4.4.2 Managing Models. Each of the Pleak components uses BPMN models as a basis. Hence, Pleak also allows to model BPMN and to share models in order to work on them collaboratively or to disseminate the analysis results.

Overall, the user has the following options for managing their models in the model listing.

- Models can be created and organized in the *My Models* tab

  - Models can be imported by first creating a new model (with the desired name for the model to be imported) and then importing the desired file when opening the new model
  - Models can be exported as *.bpmn files from the burger menu in the file listing
  - Exported models have all the Pleak specific information (e.g. scripts and data specifications) included
- Clicking on the model name opens the BPMN modeler
- Pleak editors (for adding analyzer specific information and accessing the analyzers) can be opened from the burger menu in the model line
- Models can be shared with other Pleak users using Share option
  - Models shared with the user appear under the Shared models tab
- A public link to the model can be created with the Publish option in the burger menu
  - It is possible to run the analysis on the published models but not possible to change the model when accessing it via the public link
  - This option can be used to distribute Pleak analysis results

4.4.3 Usage Restrictions. Pleak offers a range of analysis capabilities that each come with their own benefits as well as restrictions. Overall, the user can start with the visibility analysis offered by the disclosure tables that is available also for plain BPMN models. Then depending on the process it can either be enhanced with PETs (PE-BPMN stereotypes) or the operations of the tasks can be specified with the computation script. Specifying the computations allows qualitative leakage analysis with the leaks-when analyzers. Leaks-When analyzers summarize the data processing to highlight dependencies between generated data objects and the inputs of the process. Finally, if something is also known about the input data and the workflow computes an aggregation (can be also an intermediate step for guessing advantage with collaboration models) then sensitivity analyzers can be used to quantify the leakage. Note that not all analyzers may be applicable to all processes.

Pleak allows to model the BPMN models supporting the most common elements. However, different components can use different subsets of these elements. In the following we list the main guidelines for BPMN modelign in Pleak.

- Each task does one action (rule of thumb: if your task description contains the word *and* then it may confuse the analyzer). The main actions are as follows:
  - Sending or receiving data (receiving is usually done with message catch event not a task)
  - Protect or Open step for some privacy technology
  - Data processing, either public data or privacy enhanced data processing
- Each task should have its input and output data clearly marked with incoming and outgoing data associations respectively
  - For most analyzers only data processing tasks that have some inputs and output(s) are meaningful, the main exceptions are the sending tasks and for BPMN leaks-when the task before the exclusive gateway (used to specify the condition of the gateway)
- For message flows sending data we assume that the sending task has as input the data that is sent over the network and the receiving task or message catch event has the same data as output
  - Sending task should have no output data object
  - Data processing and sending tasks need to be separated, the data to be sent should be assembled by previous tasks
  - All data communication between pools should occur via the message flows as specified before
  - In short, all communication should be represented as generalized in this Pleak model: `https://pleak.io/pe-bpmn-editor/viewer/UF6lS4kSJoVNyY44SMYQ/`

- One pool should have one process (e.g. one start event)
- For analyzers supporting branching the default flow should be marked when branching
- If two data objects have the same name then they are considered to be the same data by the analyzers
- Each starting parallel gateway should be concluded with an end parallel gateway
- Some analyzers may not support spaces in the model names. Including the *(copy)* word added when you copy a model.
- Pools, lanes and data objects should have unique names
- Avoid starting data object names with the word *entity* since it may cause errors when parsing the xml of the model file.

Our wiki in `pleak.io/wiki` documents different restrictions in detail and is updated as the tools are developed further.

4.5 Collaborations

The Brandeis program was structured in a way that required collaboration between different performers. Different projects, each targeting one of the four technical areas (TA) of the project (TA1 — privacy-enhancing technologies; TA2 — human-data interaction; TA3 — integration; TA4 — measurement), were combined as Collaborative Research Teams (CRT) around the TA3 projects, with other projects potentially participating in more than one CRT. The collaboration was considered particularly significant for TA4 projects, one of which was NAPLES. We worked in all three CRTs of Brandeis, using our tools to describe, characterize, and analyse the technologies of other partners, as well as the integration platforms and activities. Below, we report the models that we created, and the analyses we ran, where analysed systems were proposed in the CRTs.

4.5.1 Mobile CRT. In our modeling and analysis we considered the development of the RapidGather [69] mobile application and its derivatives for meeting scheduling. Some of these scenarios are already discussed in Sec. 4.1.1 to discuss the use of PE-BPMN analysis. In this section we will extend this analysis of the Mobile CRT processes with some more variants of their processes.

In short the goal of the Mobile CRT was to build a privacy preserving mobile application. Overall they considered two main scenarios: crisis discovery and meeting scheduling. Both of these were approached using various privacy preserving technologies like secure multiparty computation, secure hardware (SGX) and differential privacy.

4.5.1.1 Scheduling Scenario. This section describes the BPMN Leaks-When analysis of the Mobile CRT scheduling model. In this scenario a meeting time is agreed on using the private data from the participants calendars. The process model can be accessed in `https://pleak.io/app/#/view/SE2R2wr1S49b_rbaHLhp`. You can switch between PE-BPMN and BPMN leaks-when view to see the relevant input data (privacy technologies in PE-BPMN view and computation scripts in BPMN leaks-when).

This scenario has two roles (meeting initiator and participant). Meeting initiator is a user who creates a scheduling request in order to find a suitable meeting time, taking into account each participant's calendar. Meeting initiator adds other needed participants by using preloaded "friends list" located on initiator's device. After meeting invitation is constructed by the initiator, it is sent to the Scheduling Server. In addition, the initiator sends its availability data as well, which is encrypted with MPC Server's public key. Scheduling Coordinator, which is located inside the Scheduling Server splits the meeting invitation into information (incl. meeting ID) and participant list so that other participants would only see the meeting description. The server needs the participant list so that it can send the information to the relevant users and the meeting ID is needed to be sure which meeting scheduling is taking place.

Participants must decide, whether they accept the scheduling request or not (modeled as a predicate is_userChoosestoAccept). If they do, they send a separate response to the Scheduling Coordinator and their encrypted availability data to Data Relay. If they do not accept the request, they send only their cancellation to the Coordinator. After collecting all answers, Coordinator analyses, whether all of those were positive (modeled as a predicate is_answer_positive). If they were not, the whole scheduling process is canceled and a notification is sent to relevant users. If all answers are positive, Coordinator notifies Data Relay to send the collected data to Stealth Server.

Data Relay, located inside the Scheduling Server is responsible for collecting availability data. Each received data is stored in schedule blobs. After receiving participants blob, Data Relay will check, whether all participants have sent their blob (modeled as a predicate is_blob_count_equal). Upon request from the Scheduling Coordinator, the blob is sent to the Technology-Specific Front End (in this case, we call it Stealth's FE and it is using secure multiparty computation). The blob, consisting of encrypted availability information data objects, is once again encrypted in the Front End and then sent to the Stealth's Server. Stealth Server computes the most suitable time for all participants (including the initiator) (modeled as filter_compute_meeting_time) and sends the output back to Stealth's FE, where it is converted into a meeting time object. This is sent to all participants (including the initiator).

All devices (initator's and participator's) receive a result and their device will decide how to proceed with the result, depending on its header (modeled as a predicate is_result). If it is a cancellation notification, they receive a cancellation information. If it is a meeting time, then their device's iCalendar App will make schedule an appointment in their calendars.

It is important to note, that data is always exchanged by using a Secure Channel, hence, the message content does not leak to the network observers.

BPMN Leaks-When Analysis. The output of BPMN leaks-when analysis on the scheduling process is given in Figure 56. Analysis detects 6 input objects and 1 MPC key (public key is used for encrypting from the user/Scheduling Server side and the private key is located inside the MPC Server). In addition, 7 output objects are defined in the process. In reality this process has only the result as an output but we have included the things sent over the network as well.

"Never" means that information from input never leaks into this output. "If" means that it leaks when specific conditions hold (defined, when hovering the cursor on a "if" condition) and explained in the following paragraphs. "Always" means that information from input is certainly used to compute the output.

Firstly, look at the cancellation_result, which is delivered when there is at least one user who has declined the scheduling process. Scheduling Server sends the cancellation result to all users who were part of the scheduling process. "If" condition there shows that it leaks when "is_userChoosestoAccept is passed". Is_userChoosestoAccept is the part, where participant decides, whether to participate in the scheduling process or not – if it is passed, the process can continue and when the second condition holds. This means that information from the invitation leaks, if someone has accepted and now someone decides to decline. Cancellation is sent by using the correct meeting id, which is stated in the invitation.

For positive result, "is_userChoosestoAccept is passed" (for invitation.info) is considered in addition is_blob_count_equal has to be also passed. What it means for meeting ID data field leakage, is that when a positive result is issued, it can be known, which meeting this result belongs to. Both user's and initiator's availability is leaked for positive result if "filter_compute_meeting_time has passed and is_blob_count_equal holds and is_userChoosestoAccept is passed" meaning that their availability information is used to do such computations as computing meeting time, making sure that all blobs are received when all user has accepted the scheduling. However, also indicating that only the time that is suitable for everyone is filtered out and other data about the calendar does not leak.

For meeting invitation, received from the initiator, both participants and information data fields leak since those are part of the invitation. Invitation.info leaks for received_participant_blob when user has chosen to continue with the scheduling process.

	Master_randomness_for_mpc-key.rnd	add_friend.friends	initiator_calendar_data_mobile.availability	invitation.info	old_answers.answers	participant_calendar_data_mobile.availability	schedule_blobs_A.blob
cancellation_result.meetingID	never	never	never	if	never	never	never
encrypted_package.data	never	never	never	never	never	never	never
meeting_information_for_participant.info	never	never	never	always	never	never	never
positive_result.meetingID	never	never	never	if	never	never	never
positive_result.time	never	never	if	never	never	if	never
received_initiator_blob.availability	never	never	never	never	never	never	never
received_meeting_invitation.info	never	never	never	always	never	never	never
received_meeting_invitation.participants	never	always	never	never	never	never	never
received_participant_blob.availability	never	never	never	never	never	never	never

Figure 56: BPMN Leaks-When Analysis Result for the Mobile CRT Scheduling Scenario

Since Secure Channels are used, there are no other public messages that a third-party would be able to overhear.

From this analysis it can be concluded that this way of scheduling a meeting protects the user's data.

4.5.1.2 Crisis Movement Heatmap. This section describes the collection of movement data during an emergency. The goal is to collect movement data from mobile devices to discover if a serious crisis has occurred. The core idea is that in case of an emergency like an explosion or shooting, a lot of people will be trying to get out of the danger zone. Hence, the movement data can be used to discover the occurrence and location of the crisis.

Stakeholders. This section considers all primary stakeholders in the RapidGather scenario. In essence, these can be divided to two, ones that are natural in emergency response mobile application scenario and the ones required by the privacy-enhancing technologies used in RapidGather.

- Mobile device is a device belonging to mobile device user that has RapidGather application. The following sub-sections list the relevant components of the mobile device that are supplied by different parties and therefore have to be treated as different trust-domains in privacy analysis.
- RapidGather application is a mobile device application for emergency response. It gathers the capabilities offered by the privacy technologies into one application and allows the user to specify how their data can be used.
- Privacy-Enhanced Android (PE Android) is a special version of the Android operating system that provides secure means to process sensitive data. It can use different Privacy abstraction layer modules (PAL modules) to process various kinds of data. PE Android also provides logging services to aid auditing.
- Privacy manager module is responsible for handling the permissions of applications. This includes recording the choices user makes and enforcing them when the application requires some data.

- DataCapsule PAL module is the PAL module corresponding to Helio technology. It takes input data and creates an encrypted capsule that contains the data and usage policy for the data. The information in data capsules can only be accessed using SGX technology.
- SplitLocation PAL module is used to process location data into secret shared location data that can be given to PULSAR compute servers.
- Mobile Device user is the human that has a mobile phone and is using RapidGather application. The user can make choices about how his private data can be accessed and used by the application.
- Command Center is the first responder tool used in case of emergencies. This is used by law enforcement or emergency and crisis responders to collect information about people and potential suspects in the area.
- Query Interface is the tool that the users of Command Center (Command Center operators) can use to initiate data collection or allowed analysis. It can also be used to change the alert level.
- Command Center administrators are responsible for maintaining the essential servers of the system. These components themselves are supplied by different parties.

  Communication Server is responsible for communicating with the mobile device and computing servers. Acts in a way as a router between the mobile device, computing servers, and Command Center software.

  Query Server is responsible for mapping queries of the query interface into queries of the required privacy technology and making them understandable for the plugins of the privacy technologies.

  PULSAR Plugin is the part of the Command Center that communicates with the PULSAR compute servers. It translates the queries into secure computation commands and is able to restore the result given by the compute servers.

PE-BPMN. The process model for this usecase is available in `https://pleak.io/app/#/view/xy9M9QGxehBbPAkoXSQ8`. The movement data is collected on the phone and split to two shares by SplitLocation PAL module. The aggregation of the movement is carried out by the PULSAR compute servers using secure multiparty computation and the result is displayed at the command center for inspection. The movement data is considered to be private data of the mobile device user.

The simple disclosure report in Table 22 shows that final heatmap is given only to the command center and that the intermediate results are protected from the PULSAR services. Hence, this process achieves its privacy goals.

4.5.2 Enterprise CRT. Enterprise CRT works with a number of scenarios related to the management of fleets and resources in a coalition setting, meaning that there are several parties that are willing to cooperate with each other, but do not want to share all of their data with each other.

4.5.2.1 Aid Distribution. In the aid distribution scenario, one country has asked another country to deliver aid to the first country's ports by a certain deadline. The aid is delivered on ships, which may or may not have the type of aid that is requested, which may or may not arrive at the ports of the first country by the given deadline, and which may or may not fit into the ports. Also, the berths in the ports may already have been scheduled to host other ships, and thus not be available for the ships of the second country.

In general the scenario consists of three major steps: *A* to identify *reachable ports*, *B* to determine *feasible ports*, and *C* to determine the docking slot in the port for the ship. For a ship the list of reachable ports is the set of existing ports that it can make in the given deadline. The set of feasible ports is the set of ports that are reachable by the ship that could accommodate the ship. Finally, the docking berth is chosen based on the free docks in feasible ports and the size of the ship. The scenario is specified as SQL workflow and also a simplified PE-BPMN model.

Table 22: Simple Disclosure Table for Location Heatmap

	additive heatmap shares 1	additive heatmap shares 2	current location	evaluation point	existing heatmap shares	heatmap	heatmap update 1	heatmap update 2	location sensor data	location share 1	location share 2	movement Heatmap Request	queried heatmap shares 1	queried heatmap shares 2
Administration	-	-	-	-	-	V	-	-	-	-	-	V	H	H
PE Android	-	-	V	-	-	-	-	-	V	-	-	-	-	-
Compute server 1	H	-	-	V	V	-	H	-	-	H	-	V	H	-
Compute server 2	-	H	-	V	V	-	-	H	-	-	H	V	-	H
Query Interface	-	-	-	-	-	V	-	-	-	-	-	V	-	-
RapidGather App	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SplitLocation PAL	-	-	V	-	-	-	-	-	-	H	H	-	-	-
Shared over	-	-	-	-	-	-	-	-	-	S	S	MF	S	S

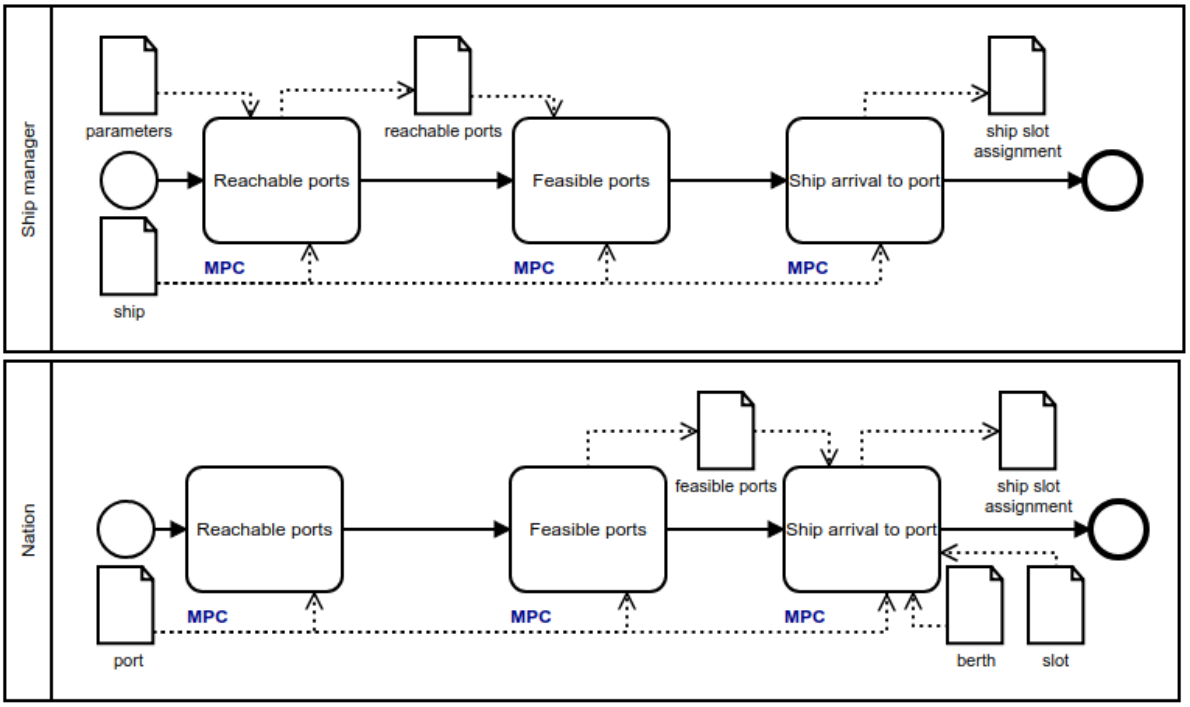


Figure 57: PE-BPMN Model for the Aid Distribution Scenario

Simple disclosure. The simplified PE-BPMN model for the aid distribution scenario considers the case where the ship and the nation use secure multiparty computation to fix details about the ship arrival to port as seen in `https://pleak.io/app/#/view/MjXG_kn9ms9MbZh-M8md` and Fig. 57. It stresses that the ship should learn which ports it can access and the nation learns the intermediate result about the feasible ports, whereas they both receive the output about where the ship is actually bound to go. This is also stressed by the simple disclosure table in Table 23. The simple data dependency graph is represented on Table, but the details of the dependency are discussed in the following sections.

Table 23: Simple Disclosure Report for the Aid Distribution Scenario

	berth	feasible ports	parameters	port	reachable ports	ship	ship slot assignment	slot
Nation	V	V	-	V	-	-	V	V
Ship manager	-	-	V	-	V	V	V	-
Shared over	-	-	-	-	-	-	-	-

Table 24: Simple Data Dependency for the Aid Distribution Scenario

	berth	feasible ports	parameters	port	reachable ports	ship	ship slot assignment	slot
berth	*	-	-	-	-	-	-	-
feasible ports	-	*	I	D	D	D	-	-
parameters	-	-	*	-	-	-	-	-
port	-	-	-	*	-	-	-	-
reachable ports	-	-	D	D	*	D	-	-
ship	-	-	-	-	-	*	-	-
ship slot assignment	D	D	I	D	I	D	*	D
slot	-	-	-	-	-	-	-	*

Leaks-when. We have modeled the scenario in Pleak, it is available in `https://pleak.io/app/#/view/cP1p11c7Q5_UjaAZS1jE` and in Fig. 58. We see that it is a workflow consisting of three steps, each of them producing a dataset. The first step computes, which ports a ship with a specific name can reach by the deadline. The second step filters from these ports the ones into which the ship does not fit. The third, most complex step selects the ports and berths that are free to receive the ship.

The leaks-when result, when applied to the first step of the workflow, is shown in Fig. 59–60. For each column (of which there are two — the ID of the reachable port, and the arrival time for the ship with given name) in the resulting dataset, we obtain a separate result, describing what inputs have affected the values in this column, how, and when.

Looking at Fig. 59, we see that the ID of the port leaks, as this is the 1st argument of the final FILTER-node. The 2nd argument describes the condition when it leaks. The leaking condition involves the attributes `longitude` and `latitude` of the same row that has the ID of the port that leaks. It also involves the only row of the `parameters`-table. Moreover, the condition states that there must exist a ship, such that

- the name of the ship is equal to the given `shipname`;
- the location and the maximum speed of the ship must be such, that it can reach the port by the given deadline. Here the operation Δ computes the distance between two points, given by their geographic coordinates.

Fig. 60 tells us that an expression involving the attributes in the rows of the `ship`- and `port`-tables does leak. We have to understand that this is the arrival time of the ship.

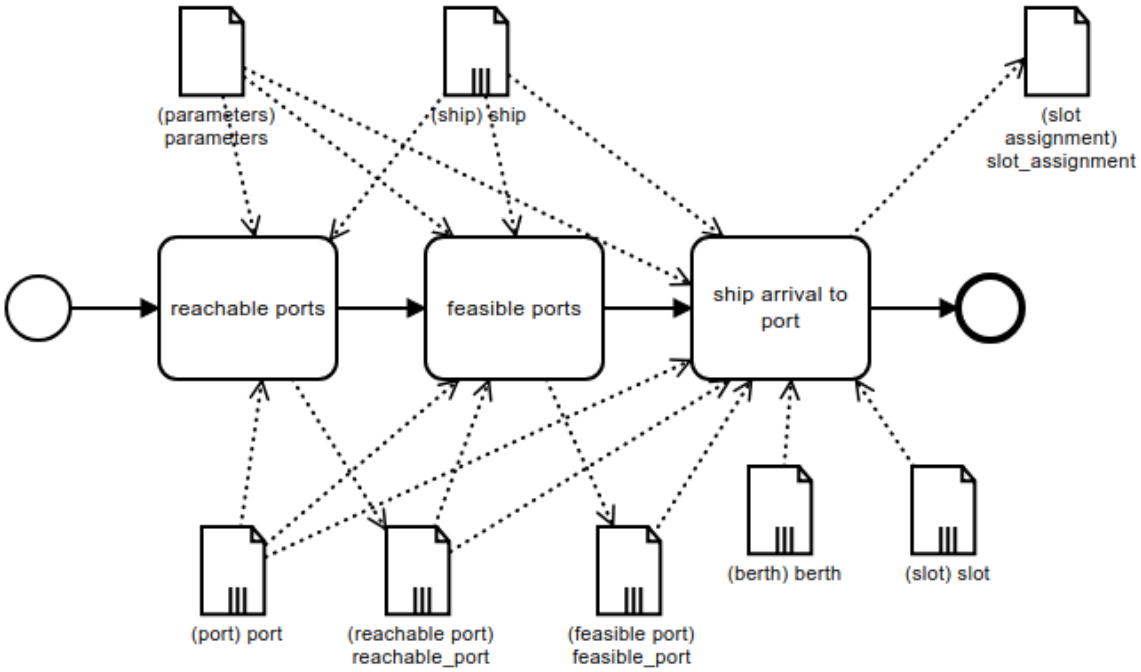


Figure 58: Workflow for the Aid Distribution Scenario

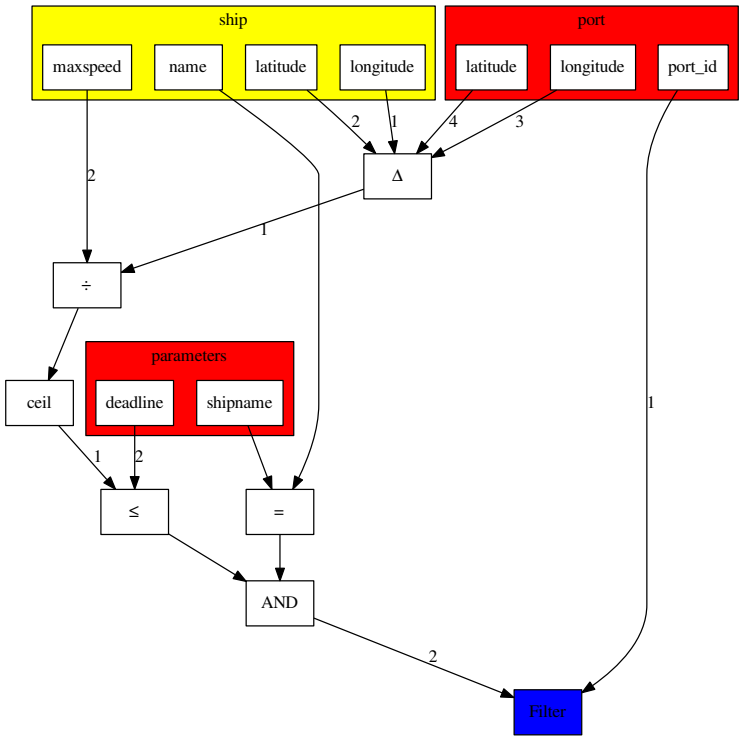


Figure 59: Leaks From the First Step of the Aid Distribution Scenario (1st Part)

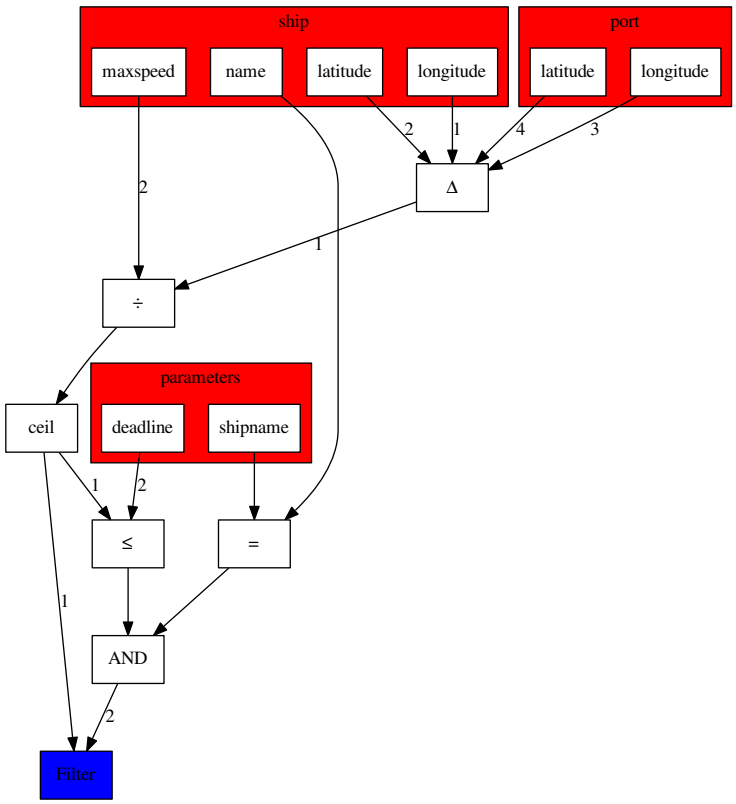


Figure 60: Leaks From the First Step of the Aid Distribution Scenario (2nd Part)

The leaks-when result, when applied to the second step of the workflow, is given in Fig. 61. As the resulting dataset only has a single column, there is a single result. We again see that the ID of the port leaks. The leakage condition is more interesting. We see that there have to be two rows in the `ship`-table (which are allowed to coincide), such that one row describes a ship that can reach the port with the leaking ID by the given deadline, and the second row describes a ship that can fit into this port. Moreover, both ships must have their name equal to `shipname` given as a parameter.

There are a total of seven results for the third step of the workflow, corresponding to the four columns of the dataset resulting from this step. There can be more than one result for each column, because internally the analyzer distinguishes the existence and non-existence of a matching row when performing outer joins. All these results are much more complex than the previous ones. Fig. 62 contains one of these results. Due to such complicated results, we believe that the for larger systems, the leaks-when analyzer should be used to state, whether some important checks have to be made for the leak to be possible, or whether the leakage can only occur through certain sanitizing operations. But it makes no sense to show to the user the entire graph, corresponding to the entire computation that the system does.

Sensitivity. The somewhat simplified example model is available in `https://pleak.io/app/#/view/ROM40oEkPtkE7uQ6pNnk` and Fig. 58.

It contains the following SQL statements.

```
CREATE TABLE parameters (
  param_id INT8 PRIMARY KEY,
  deadline INT8,
  shipname TEXT);

```

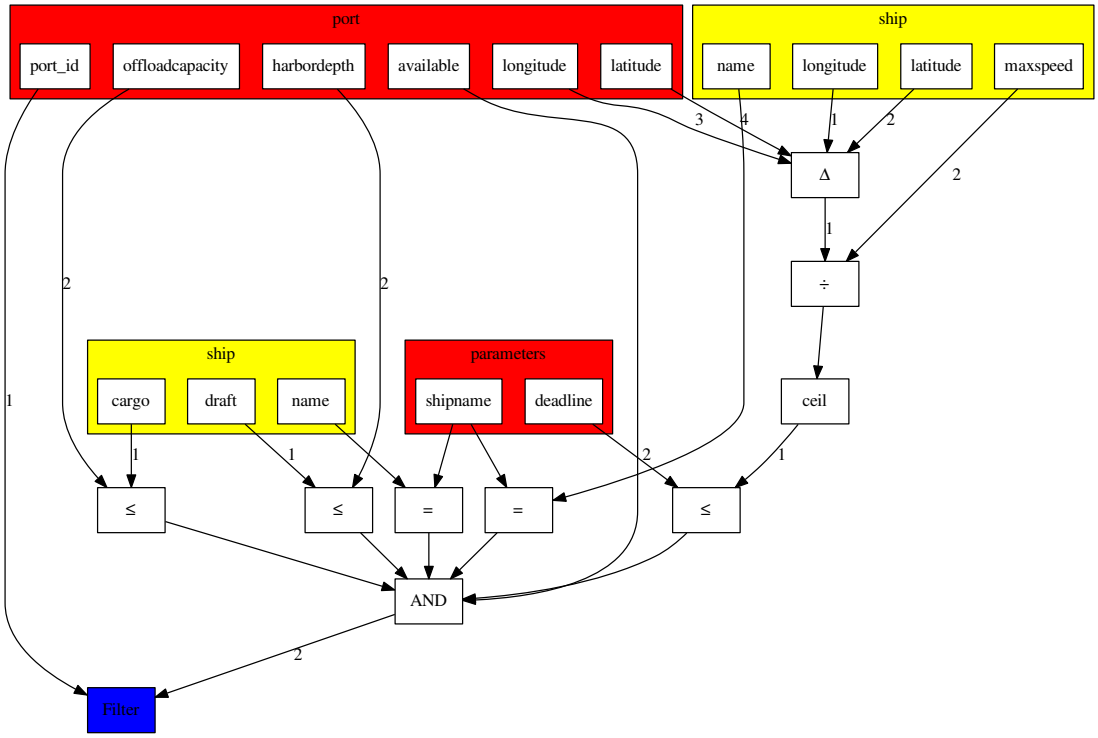


Figure 61: Leaks From the Second Step of the Aid Distribution Scenario

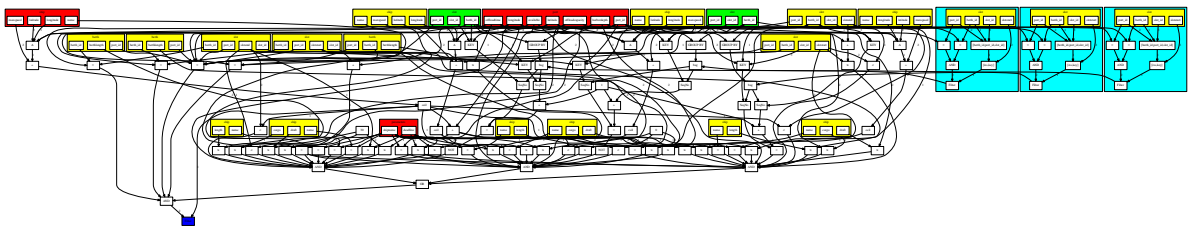


Figure 62: Leaks From the Third Step of the Aid Distribution Scenario (One Out of Seven Parts)

```
6  create table port (
7    port_id INT8 primary key,
8    name TEXT,
9    latitude INT8,
10   longitude INT8,
11   offloadcapacity INT8,
12    offloadtime INT8,
13    harbordepth INT8,
14    available Bool);
15
16 create table ship (
17   ship_id INT8 primary key,
18   name TEXT,
19   cargo INT8,
20   latitude INT8,
21   longitude INT8,
22   length INT8,
23   draft INT8,
24   max_speed INT8
25 );
26
27 create or replace function reachable_port() returns TABLE (
```

```
    port_id INT8,
    arrival FLOAT8
) as $$
SELECT
    port.port_id AS port_id,
    ((ship.latitude - port.latitude) ^ 2 +
     (ship.longitude - port.longitude) ^ 2) ^ 0.5 / ship.max_speed AS arrival
FROM port, ship, parameters
WHERE
     ship.name = parameters.shipname
$$ language SQL;

create or replace function feasible_port()
returns TABLE (
    port_id INT8
) as $$
SELECT
    port.port_id AS port_id
FROM reachable_port, port, ship, parameters
WHERE
    reachable_port.port_id = port.port_id
    AND port.available
    AND port.harbordepth >= ship.draft
    AND port.offloadcapacity >= ship.cargo
    AND ship.name = parameters.shipname
$$ language SQL;

create or replace function ship_arrival_to_port()
returns TABLE (
  min_time FLOAT8
) as $$
SELECT min(rport.arrival)
FROM
    reachable_port AS rport,
    feasible_port AS fport,
    port, slot, berth, ship, parameters
WHERE
    port.port_id = fport.port_id
    AND port.port_id = rport.port_id
    AND port.port_id = berth.port_id
    AND slot.port_id = berth.port_id
    AND slot.berth_id = berth.berth_id
    AND ship.name = parameters.shipname
    AND berth.berthlength >= ship.length
    AND slot.slotstart <= parameters.deadline
    AND slot.slotstart + port.offloadtime <= slot.slotend
$$ language SQL;
```

Table 25 gives the global sensitivity analysis results on the aid distribution scenario. It is interesting to note that the sensitivity of both reachable and feasible port tasks is infinite with respect to each of their inputs. On the other hand, the final task actually has only sensitivity 1 with respect to each of its inputs.The given table shows the aggregated sensitivities over the whole workflow. For example, we can see in Table 25 that the berth and slot values do not affect the reachable or feasible port data, which is clear because they are indeed not used to compute these values. For example, this is easy to see also from the simple data dependency result in Table 24. On the other hand, we can see that the workflow has infinite sensitivity also for some of the data affecting the slot assignment that is derived from the fact that these were infinitely sensitive for the intermediate results used to compute the slot assignment. Hence, this adds more knowledge for the cases when there is some dependency between data objects.

4.5.2.2 Pandemic. Pandemic scenario is related to counting how many people have a particular disease status. The process model for this usecase is available in `https://pleak.io/app/#/view/gYmOt3NhmdzetD7urJJ_` and in Fig. 63. There are four abstract labels 'S', 'I', 'R', 'D', each denoting some state. The analyst wants to learn a histogram, which shows how the counts are distributed in different groups (so-called *communities*). The output needs to be ϵ-differentially private, protecting a single person's disease status. We want to give a meaning to ϵ, reducing it to the probability of guessing particular person's disease status.

Table 25: Global Sensitivity for the Aid Distribution Scenario

	reachable port	feasible port	slot assignment
parameters	Infinity	Infinity	Infinity
ship	Infinity	Infinity	Infinity
port	Infinity	Infinity	Infinity
berth	0	0	1
slot	0	0	1

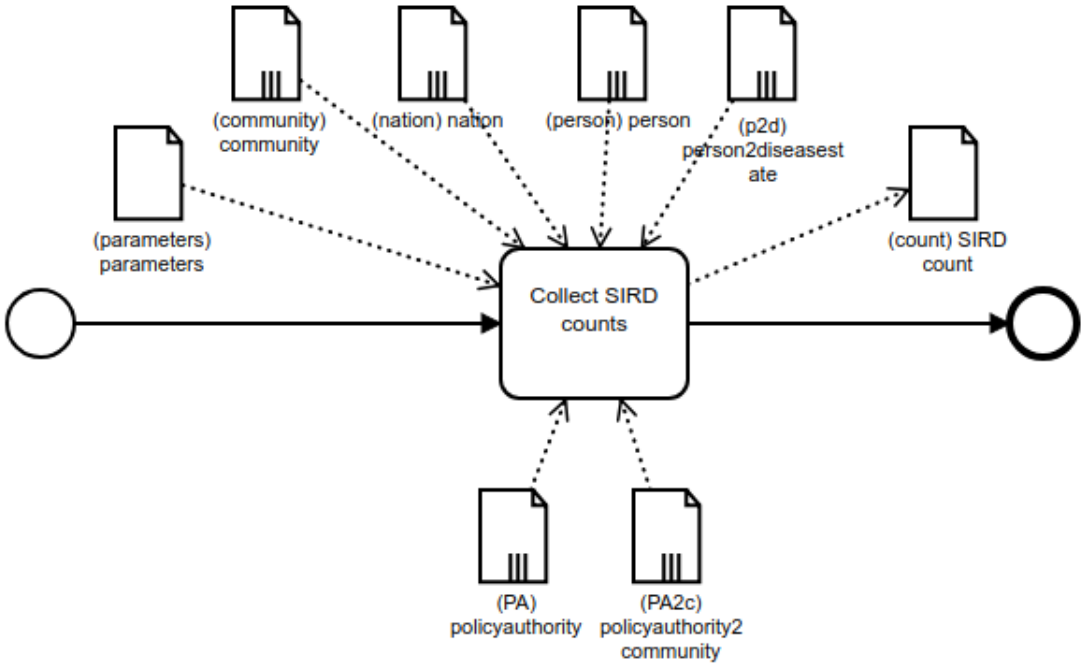


Figure 63: Data Processing Workflow for the Pandemic Scenario

To estimate the advantage on guessing probability, we use results of Sec. 3.3.10.1. Initially, the attacker has no knowledge about any person's status, and we may assume that the prior probability of guessing a state is 0.25. The values themselves are discrete and the distance between any two of them is defined as 1. We want to know how likely the attacker guesses the count precisely.

Let p be the prior probability of attacker guessing the disease state correctly. Using results of Sec. 3.3.10.1, given the bound on advantage ϵ', we compute differential privacy parameter ϵ as

$$\epsilon = -\ln\left(\frac{p}{1-p} \cdot \left(\frac{1}{\epsilon' + p} - 1\right)\right) \tag{45}$$

In the case of counting histogram, we know that each person occurs in at most one of the groups. Hence, the queries $q_1(x), \ldots, q_m(x)$ that output the histogram bars can be written as $q_1(x_1), \ldots, q_m(x_m)$, where x_i is the subset of those people who belong to the i-th group. Since we are looking for the probability of guessing status of a single person, we may use ℓ_1-norm to combine $x_1, \ldots, x_n$, since the change in person's status would affect only one of the x_i (we assume that different people are not correlated). This gives differential privacy for the entire output $\epsilon = \max_{i \in [n]} \epsilon_i$, where ϵ_i is the value that we would have if only the i-th histogram bar had been output. Hence, after computing ϵ as in Equation 45, we can take $\epsilon_i = \epsilon$ when sampling the additive noise for a single output.

It is easy to estimate the sensitivity c w.r.t. disease state: if the label of some person changes, the count of a histogram may at most change by 1. In this particular case, the local sensitivity is no different

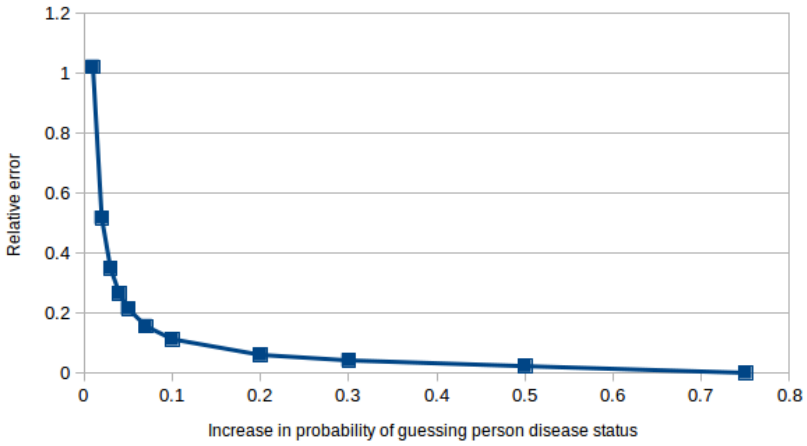


Figure 64: Privacy vs Relative Error for Distribution of Counts per Nation

from global sensitivity. We compute the scaling parameter as described in Sec. 3.3.8.2, which is $\frac{c}{b} = \frac{1}{b}$ for $b = \frac{\epsilon}{\gamma+1} - \beta$, where γ is a noise distribution parameter, and β is a smoothness parameter. Since global sensitivity is no different from local in this case, we have $\beta = 0$. In our tool, we use $\gamma = 4$. Hence, the scaling parameter is $\frac{\gamma+1}{\epsilon} = \frac{5}{\epsilon}$ for the ϵ computed as above. The noise magnitude can now be estimated as in Sec. 3.3.8.5. For Cauchy distribution, the noise magnitude $a = \frac{5}{\epsilon}$ means "with $\approx 78\%$ probability the noise is below a".

We have run the analysis on a smaller dataset with 5000 people, and plotted some graphs demonstrating privacy vs relative error for several queries. First, let us assume the attacker has no knowledge about any person's status, and hence the prior probability of guessing a state is $p = 0.25$ for each of the four disease states.

Distribution of counts per nation. The query returns a histogram, showing how many people have a certain disease status in each of the three nations.

```
SELECT nation.nation_name, person2diseasestate.diseasestate, COUNT(*)
FROM person
        JOIN community ON community.community_id = person.residence
        JOIN person2diseasestate ON person2diseasestate.person_id = person.person_id
        JOIN nation ON nation.nation_id = community.nation_id
WHERE person2diseasestate.transitionDate <= '04-10-2017'
GROUP BY nation.nation_name, person2diseasestate.diseasestate
ORDER BY nation.nation_name;
```

The analysis results are depicted in Figure 64. We see that we can keep both the relative error and the advantage below ca. 10%.

Distribution of counts per community. The query returns a histogram, showing how many people have a certain disease status in each community group. The partitioning is now finer, hence, we need more noise to conceal the data.

```
SELECT community.community_name, person2diseasestate.diseasestate, COUNT(*)
FROM person
        JOIN community ON community.community_id = person.residence
        JOIN person2diseasestate ON person2diseasestate.person_id = person.person_id
WHERE person2diseasestate.transitionDate <= '04-10-2017'
GROUP BY community.community_name, person2diseasestate.diseasestate
ORDER BY community.community_name;
```

The analysis results are depicted in Figure 65. We see that we can now keep the relative error and the advantage below 20%.

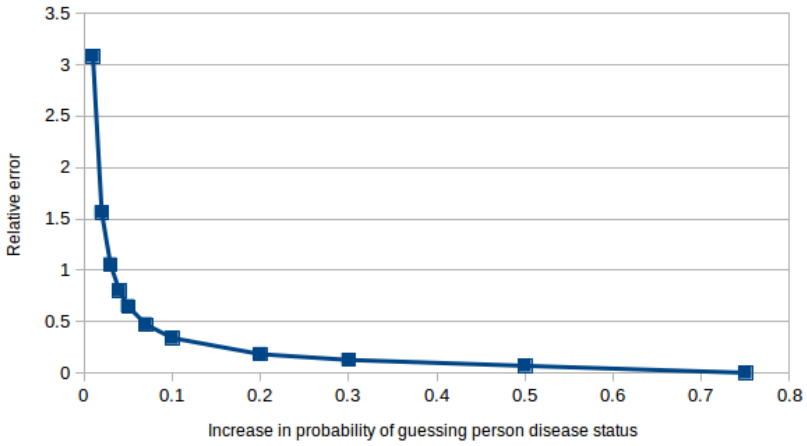


Figure 65: Privacy vs Relative Error for Distribution of Counts per Community

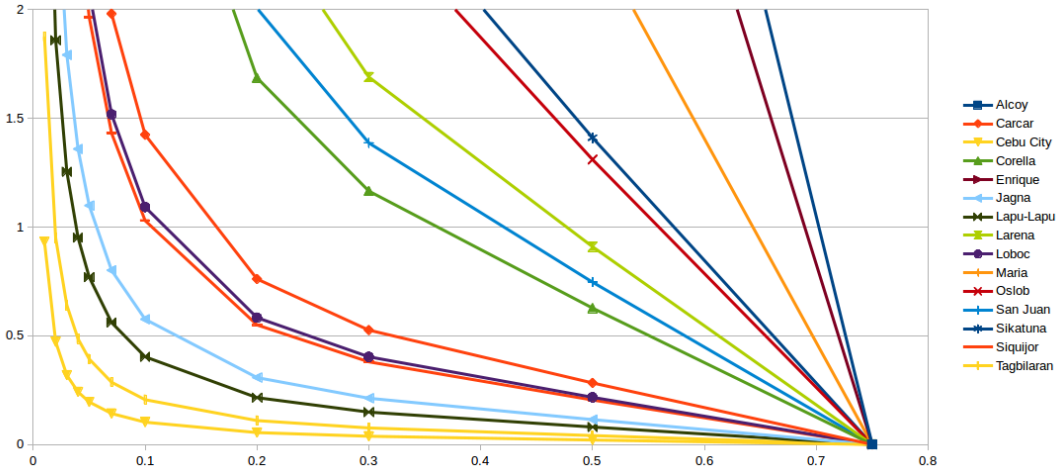


Figure 66: Communitywise privacy vs relative error for distribution of counts per community

Policy restricted distribution of counts per community. This set of queries is similar to the previous one, but it outputs a histogram bar only if the query maker has the right to see it. In our dataset, each policy authority is allowed to see the count of exactly one community. Here is an example of this query for authority `'AlcoyAuthority'`.

```
SELECT community.community_name, person2diseasestate.diseasestate, COUNT(*)
FROM person
JOIN community ON community.community_id = person.residence
JOIN person2diseasestate ON person2diseasestate.person_id = person.person_id
JOIN policyauthority2community ON
        policyauthority2community.community_id = community.community_id
JOIN policyauthority ON
        policyauthority.authority_id = policyauthority2community.authority_id
WHERE policyauthority.authority = 'AlcoyAuthority'
    AND person2diseasestate.transitionDate <= '04-10-2017'
GROUP BY community.community_name, person2diseasestate.diseasestate
ORDER BY community.community_name;
```

The results for all policy authorities are depicted in Figure 66. We see that the results are very different for different communities. While `'Cebu City'` enjoys the possibility of error and advantage below 10%, the situation is awful for `'Alcoy'`. The reason is that the population of `'Cebu City'` is very large(1118 people), but it is very very small for `'Alcoy'` (4 people), so the relative error becomes naturally huge.

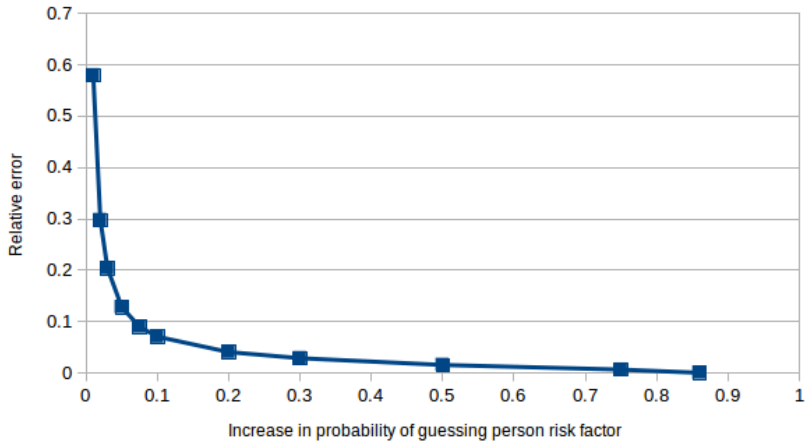


Figure 67: Privacy vs Relative Error for Distribution of Disease Risk Factor Counts for the Entire Population

Counting disease risk factor. The set of Pandemic queries contains one query that returns a histogram, showing how many people have a certain disease risk factor. The difference from the disease state counts is that there are now 7 possible values, and the ordering seems important, so we have $R = 7$ in the estimation of ϵ. The sensitivity if this query w.r.t. the disease risk factor is 1, similarly to counting disease states. The counts are now done for the entire population, without partitioning it to nations or communities. We again assume that the prior distribution is uniform, so $p = \frac{1}{7} \approx 0.143$.

```
SELECT person2diseaseriskfactor.riskfactor_id, COUNT(*)
FROM person2diseaseriskfactor
        JOIN person ON person.person_id = person2diseaseriskfactor.person_id
GROUP BY person2diseaseriskfactor.riskfactor_id
ORDER BY person2diseaseriskfactor.riskfactor_id;
```

The analysis results are depicted in Figure 67. We see that we can keep both the relative error and the advantage below ca. 8%. The results are better since more people are involved into the count, and they are not partitioned to nations or communities.

Non-uniform prior. Assuming $p = 0.25$ for each of the four disease states is a quite rough simplification. In reality, the attacker would be able to know e.g. that a person is more likely recovered than dead. Let us discuss another way of estimating prior distributions.

A stream process of each object can be characterized by a state transition matrix, which defines the probabilities of moving between the states. This matrix can be known in advance, or it can be computed from the data itself. For each object u, we can compute the probability of going from the state s_a to the state s_b as $\frac{N_{a\to b}}{T-1}$, where $N_{a\to b}$ is the total number of times the object moved from state s_a to state s_b, and T is the total number of time units. The computation includes also the cases $a = b$, so $T - 1$ is the correct value for the total number of transitions, even if the object has not changed its state at some time point. After constructing the transition matrix B_i for each $i \in [n]$, we may find the probabilities p^t_{ij} of the object u_i being in the state s_j at time t.

In our experiment, we compute the transition between different states of a single person. This actually does not correspond to real world, as the next state of a person depends not only on his own current state, but also on the current states of the other people, who may infect him. We would need a more complicated Markov process that takes into account the state of the entire database, but we would need more data to compute such a model.

We apply our theory to the 5000 people Pandemic dataset, where the disease states are measured for 10 different timepoints. Our analysis consists of the following steps:

1. Compute prior probabilities using transition matrix method.

2. Given the number of timepoints $T = 10$, the initial state distribution, and the desired bound δ on guessing advantage, we compute the $\epsilon_i^1, \ldots, \epsilon_i^T$ sufficient after $1, \ldots, T$ queries are made.

3. For each t, compute the relative error caused by the given choice ϵ_i^t, and plot the graphs.

We now describe how the experiment has been performed. First, we collect all possible states and times that have been recorded.

```
DROP TABLE IF EXISTS times;
SELECT DISTINCT transitiondate AS t INTO times FROM person2diseasestate
ORDER BY transitiondate ASC;

DROP TABLE IF EXISTS states;
CREATE TABLE states (s dstype);
INSERT INTO states VALUES ('S'), ('I'), ('R'), ('D');
```

First, we want to have a clear view on the disease state of each person for each day. The table `person2diseasestate` contains only transitions, and we want to recover the states of all people from that. Let us create a new table `data` for this, where `s` is the state, `u` is the object, and `t` is the timepoint.

```
CREATE TABLE data (s dstype, u bigserial, t date);
```

Initially, all the people are in state 'S', so until any transition takes place, the person is still in 'S'. A person was in recovered state on day d iff his state has changed to 'R' before (including) the day d.

```
INSERT INTO data
SELECT 'R', person.person_id, dates.transitiondate
FROM person, dates, person2diseasestate AS pds
WHERE pds.person_id = person.person_id
AND pds.diseasestate = 'R'
AND pds.transitiondate <= dates.transitiondate;
```

Similarly, a person was dead on day d if his state has changed to 'D' before (including) the day d.

```
INSERT INTO data
SELECT 'D', person.person_id, dates.transitiondate
FROM person, dates, person2diseasestate AS pds
WHERE pds.person_id = person.person_id
AND pds.diseasestate = 'D'
AND pds.transitiondate <= dates.transitiondate;
```

A person was sound on day d if his disease state has not changed before (including) the day d.

```
INSERT INTO data
SELECT 'S', person.person_id, dates.transitiondate
FROM person, dates
WHERE NOT EXISTS (SELECT * FROM person2diseasestate
                  WHERE person_id = person.person_id
                  AND transitiondate <= dates.transitiondate);
```

Finally, a person was infected on day d if his disease state has changed to 'I' before (including) the day d, and has not changed to any other state before (including) the day d.

```
INSERT INTO data
SELECT 'I', person.person_id, dates.transitiondate
FROM person, dates, person2diseasestate AS pds
WHERE pds.person_id = person.person_id
AND pds.diseasestate = 'I'
AND pds.transitiondate <= dates.transitiondate
AND NOT EXISTS (SELECT * FROM person2diseasestate
                WHERE person_id = person.person_id
                AND transitiondate <= dates.transitiondate
                AND NOT diseasestate = 'I');
```

To compute $\Pr[s_a \to s_b]$ for each pair of states, we need to compute the following queries:

```
DROP TABLE IF EXISTS transitions;
SELECT d1.s AS s1, d2.s AS s2, SUM(1 / totalcount.cnt :: FLOAT) AS pr
INTO transitions
FROM data AS d1, data AS d2,
     (SELECT d1.s, COUNT (*) AS cnt
```

```
      FROM data AS d1, data AS d2
      WHERE d1.u = d2.u
      AND d2.t - d1.t = 1
      GROUP BY d1.s) AS totalcount
WHERE d1.u = d2.u
AND d1.s = totalcount.s
AND d2.t - d1.t = 1
GROUP BY d1.s, d2.s;
```

The table `transitions` now contains all data describing a transition matrix B. We now set $\pi_0 = [1,0,0,0]$ (initially, all people are in the state 'S'), and then compute $\pi_k = \pi_{k-1} \cdot B$ for all $k \in \{1,\ldots,T\}$, where T is the number of timepoints. Let us assume that the results are put into table `priors(s dstype, t date, pr FLOAT);`

We are going to plot the results using a 2D graph, where the x-axis is the guessing advantage, and the y-axis is the relative error. For each time point, we plot a separate graph line. Since the histogram has several real outputs, and there is a separate histogram for each timepoint, each having its own relative error, we plot the ℓ_2-norm of all errors as an aggregated error. An important difference from the previous phase experiments, where only one timepoint was considered, since the prior probabilities of different timepoints are different, having same *guessing advantage* δ does not mean that the *guessing probability* is the same.

To estimate the error correctly, we need to compute the actual counts.

```
SELECT statetimes.s AS s, statetimes.t AS t, COUNT(data.u) AS cnt
INTO occupancy
FROM statetimes
LEFT JOIN data ON statetimes.s = data.s AND statetimes.t = data.t
GROUP BY statetimes.s, statetimes.t;
```

We take a set of different guessing advantages δ

```
CREATE TABLE deltas (delta FLOAT);
INSERT INTO deltas VALUES (0.01), (0.02), (0.03), (0.04), (0.05), (0.07),
                          (0.1),  (0.2),  (0.3), (0.5),  (0.75), (0.9),
                          (0.92), (0.95), (0.96), (0.97), (0.98), (0.99);
```

From prior probabilities and δ-s, we can now compute the corresponding values for ϵ. Since PostgreSQL does not support $\epsilon = \infty$, we write $\epsilon = -1$ instead, and treat is as a special case.

```
DROP TABLE IF EXISTS epsilons;
SELECT priors.s AS s, priors.t AS t, deltas.delta AS delta,
   (CASE WHEN priors.pr = 0 OR deltas.delta + priors.pr >= 1 THEN -1
   ELSE -ln(((1.0 / (deltas.delta + priors.pr)) - 1) *
   (priors.pr / (1 - priors.pr))) END) AS epsilon
INTO epsilons
FROM priors, deltas;
```

We are now ready to estimate the relative error. We know that the sensitivity of the `COUNT` query w.r.t. diseasestate is 1. Since this sensitivity is also global, the smoothness parameter is $\beta = 0$. In this experiment, we estimate the amount of noise coming from Laplace distribution $\frac{1}{2\lambda} \cdot e^{-\frac{|x|}{\lambda}}$, where $\lambda = \frac{1}{\epsilon}$. We want to find an upper bound on the noise achieved with probability $\approx 78\%$, as we did it with Cauchy distribution. The noise magnitude can be estimated as in Sec. 3.3.8.5, and $a = \frac{-\ln(1-0.78)}{\epsilon}$ is a fine bound on noise magnitude. If there are k related DP outputs, we need to divide ϵ by k to get the same privacy level, so the noise magnitudes will be of the form $\frac{-\ln(0.22)k}{\epsilon}$. Hence, we compute the error once for each ϵ, and then just multiply it by k for the k-th graph.

```
SELECT epsilons.s AS s, epsilons.t AS t, epsilons.delta AS delta,
       (CASE WHEN epsilon = -1 THEN 0 ELSE (-ln(0.22) / epsilons.epsilon) /
       occupancy.cnt END) AS error
INTO pdps
FROM epsilons, occupancy
WHERE epsilons.s = occupancy.s
AND epsilons.t = occupancy.t;
```

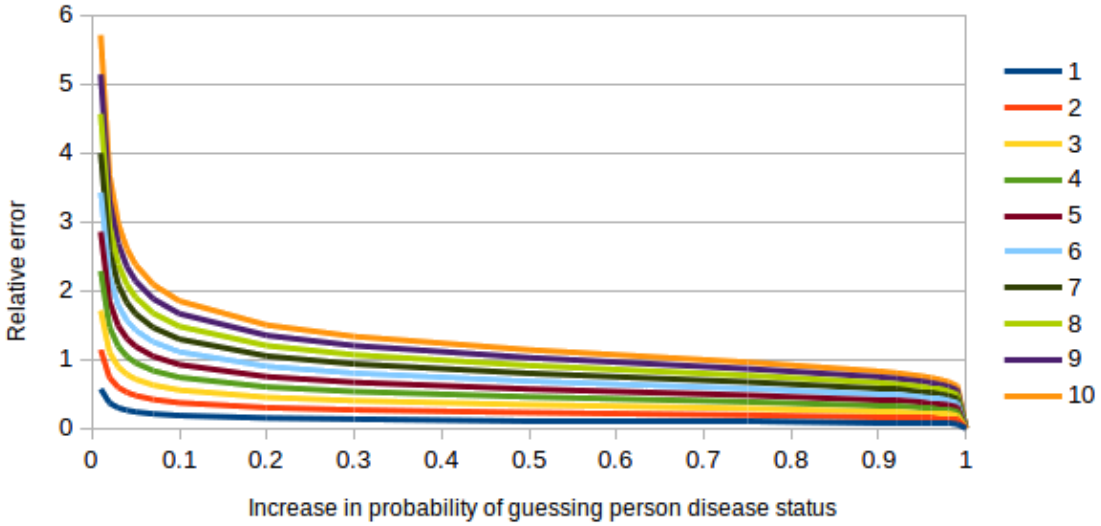


Figure 68: Privacy vs Relative Error for Pandemic Scenario with Priors Generated from Markov Process for $1 \ldots, 10$ **queries**

We have performed this experiment on Pandemic scenario with 5000 people and 10 days. We assume that the counts have been published for each day, and report how much the attacker learns if he observes $1 \ldots 10$ of these counts. The results are depicted in Fig. 68.

In a real applications, it is likely that it is not known in advance how the prior distribution changes with time. We may assume an attacker who starts with some prior, which he updates after he sees output of some query applied to that data. If the policy authority assumes that posterior probability may change up to δ after seeing T outputs, we may bound advantage of each single step by $\frac{\delta}{10}$.

```
DROP TABLE IF EXISTS epsilons;
SELECT statetimes.s AS s, statetimes.t AS t, deltas.delta AS delta,
       (CASE WHEN t * delta / 10.0 + priors.pr >= 1 THEN -1
       ELSE -ln(((1.0 / (t * delta / 10.0 + priors.pr)) - 1) *
       (((t - 1) * delta / 10.0 + priors.pr) /
       (1 - ((t - 1) * delta / 10.0 + priors.pr)))) END) AS epsilon
INTO epsilons
FROM deltas, statetimes, priors
WHERE priors.s = statetimes.s;
```

Since a prior probability 0 never becomes anything different from 0, we started from a distribution where all states have positive probabilities. For the experiment, we took $\Pr['S'] = 0.97$, and $\Pr['I'] = \Pr['R'] = \Pr['D'] = 0.01$. The results are given in Fig. 69, and we see that the error are around 10 times higher than for Fig. 68 where we used a Markov process. Indeed, updating priors step by step may lead to huge overestimations, since an upper bound is not tight.

4.5.3 IoT CRT. The Internet-of-Things CRT is fitting out a major building at the University of California, Irvine (UCI) campus with a number of sensors, including various sensors for presence (WiFi access points, beacons, cameras) and for measuring environmental conditions (in particular, temperature). The collected data can be used to let the building respond better to the needs and preferences of its inhabitants, to simplify the navigation in the building, to facilitate interaction between the inhabitants, etc. While some applications for concrete tasks have currently been built, e.g. a system for locating and reserving meeting rooms, or an app guiding its user throughout the building, the goal of IoT CRT is not to build apps for particular tasks, but to support an open API through which the data collected through the sensors may be used.

TIPPERS system processes the data collected by the sensors, as indicated by the policies of other stakeholders. It receives the raw data from the installed sensors and turns it to presence data. For this purpose, it makes use of the registry about the relationships between users and their devices; it maintains

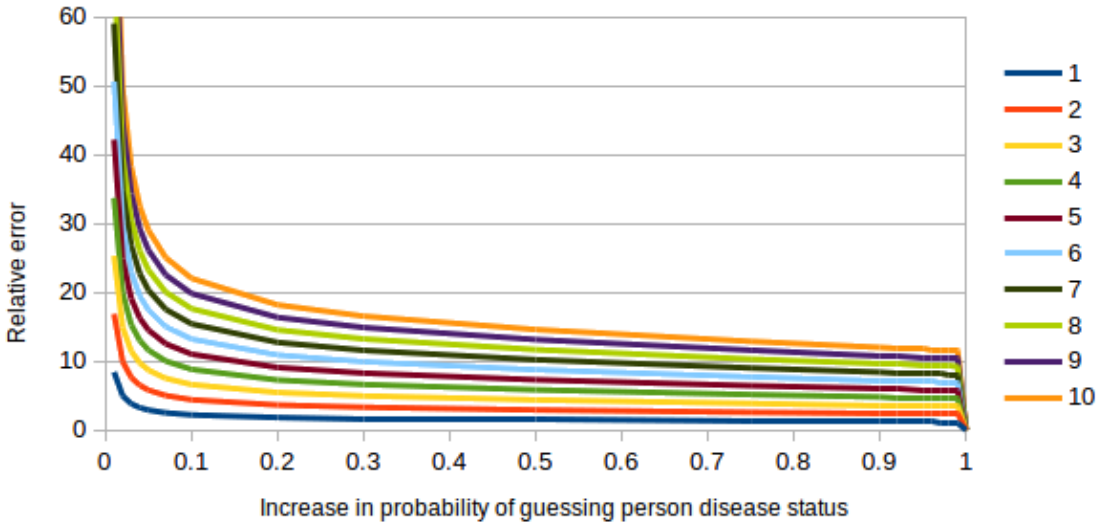


Figure 69: Privacy vs Relative Error for Pandemic Scenario with Priors Generated Iteratively for $1 \ldots, 10$ **queries**

this registry. It stores the presence data of users according to the policies of both the sensors and the users. It can respond to queries about the locations of users either in present or in the past, again subject to users' policies. I.e. the TIPPERS system is trusted in enforcing at least certain parts of the policies of the users.

TIPPERS system provides an interface to these privacy-enhancing technologies that are available in the system. Some of the data collected by the building, or given by the users, may be stored and processed with the help of these technologies, so the TIPPERS system does not necessarily have full access to all data.

It is possible to recognize a couple of well-defined subsystems of the TIPPERS system. The *location tracking subsystem* receives the locations of the mobile devices in the building, associates them with users, and stores the location data of them. The *policy enforcement subsystem* releases the location data subject to policies of the involved users. The *data publishing subsystem* makes the bulk of location data available to researchers, applying privacy enhancing technologies before its release.

The IoT scenarios are somewhat different from the ones in other two CRTs. Namely, IoT CRT "naturally" considers long-running processes, e.g. the tracking of the locations of the inhabitants of the building. These bring some unique challenges into the modeling and analysis, and have inspired certain conventions that we have applied during our modeling of the relevant processes.

4.5.3.1 Presence Scenario. In the presence scenario we consider analysis of campus locations and the number of people in there. The building is the Donald Bren Hall — the computer science building at UCI campus. It contains several subsystems for sensing, for systematizing collected data, for managing the users and their privacy expectations, for storing and updating certain (but not all) sensitive data. Users are the regular and the sporadic inhabitants of the Bren Hall. The building collects their data, which it uses for their well-being.

The building is outfitted with physical sensors collecting data about users, their mobile devices, and the environment. There are WiFi access points, which record the MAC addresses of the devices associating with them, and provide a coarse-grained location sensing. There are Bluetooth proximity sensors, which associate with nearby mobile devices and provide a more fine-grained sensing of locations. There are temperature sensors.

IoT Resource Registry keeps track of the sensors present in the building, and any privacy policies associated with them. These privacy policies describe the extent of control available to the users when

regulating the movement of data pertaining to them, collected by these sensors, through the entire TIPPERS system. The privacy policies of the users have to be aware of the policies associated with sensors.

Formally, there is a set of objects (e.g. people) and a set of possible states of these objects (e.g. locations). From time to time, an object reports its state to the system. The model consists of one table with the schema (`object_id`, `startTime`, `endTime`, `state`, `confidence`), containing measurements of an object's state in certain time periods. From this table, a state occupancy table with the schema (`state`, `timestamp`, `count`) is computed. The occupancy table tells how many objects have had certain state at a certain point of time. A histogram of state occupancy is periodically updated and shown to an observer. Let q be the query computing the occupancy counts. The output has form of a stream, and some privacy mechanism $\mathcal{M}_q(\cdot)$ is applied to each timepoint of the stream, to prevent the observer from learning the states of individual objects. Our goal is to estimate how much the attacker is able to infer about states of individual objects from the occupancy counts, comparing different privacy mechanisms with different parameters. Our goal is to answer the following questions:

1. How much the attacker is able to infer from the output *without differential privacy*?
2. How much the attacker is able to infer from the output *with ϵ-differential privacy*?
3. What is the utility loss for the given ϵ?

We do not provide a Pleak model for this use case. The model itself is very simple, and there are no Pleak analysis tools that could be applied to it directly since the settings are too specific. In the context of our privacy study, we are comparing the following privacy mechanisms.

Laplace Mechanism. Standard differential privacy (DP) mechanism, which adds noise sampled from Laplace distribution $Lap(\lambda)(z) \sim \frac{1}{2\lambda} e^{\frac{-|z|}{\lambda}}$, where λ is a scaling parameter.

PeGaSus Mechanism. PeGaSus [72] is an DP mechanism for stream data. Differently from TTL, the PeGaSus mechanism is formally well defined. After adding Laplace noise to an output, the results are smoothed, so that the stream data gets better utility while still satisfying ϵ-DP as it would with plain Laplace noise.

Time-to-Live Mechanism. In the TTL (time-to-live) mechanism, the unique identifiers of the objects change every k minutes for a fixed parameter k. Formally, its goal is to protect data *before* computing the counts, so that an observer would not be able to track movement trajectories. It is *not* a DP mechanism, and it gives no provable guarantees of privacy w.r.t. released counts, as counting does not depend on the object identities anyway. The only way in which TTL can affect released counts is that the same user may be recorded multiple times if his identity has been updated within the 10-minute span for which the count is computed.

Experiment. The experiments are performed on presence and occupancy records for 3 months (February, March, April, 2018), which comprises $N = 89$ days. We use the first $N - 1$ days of presence data for constructing prior probabilities. We then use the last N-th day of the occupancy table to compute posterior probabilities, showing how attacker's guesses improve compared to guessing from prior. Our analysis consists of the following steps.

1. We split a day into 10-min spans. This gives us $T = 144$ time units per day.
2. For each 10-min span t of a day, for each location l and each user u, we compute prior probabilities $P_{prior(u,t,l)}$ from the first $N - 1$ days.
3. Based on the prior probabilities $P_{prior(u,t,l)}$ and the noisy occupancy counts generated by a particular privacy mechanism $\mathcal{M}$ on the N-th day, we compute the posterior probabilities $P^{\mathcal{M}}_{noisyOcc(u,t,l)}$. Among other mechanisms, we estimate $\mathcal{M}(x) = x$ (guessing from true occupancy counts) and $\mathcal{M}(x) = \perp$ (guessing just from prior).
4. Let $P_{true(u,t,l)} \in \{0, 1\}$ be the actual user locations, i.e., $P_{true(u,t,l)} = 1$ iff u was in location l at time t. Compute the following for each user u and time t:

$$P^{\mathcal{M}}_{guess(u,t)} = \sum_{l \in \mathcal{L}} P^{\mathcal{M}}_{noisyOcc(u,t,l)} \cdot P_{true(u,t,l)} \quad .$$

5. Plot aggregate privacy metric: how many people have been localized correctly from $P^{\mathcal{M}}_{guess(u,t)}$ with a probability within certain range, excluding those who have been localized with similar confidence purely from prior.

For a fixed timepoint t, the attacker receives a prior distribution of location of an "average user", expressed as $p^t_l \in [0..1]$ for all $l \in \mathcal{L}$, where $\sum_{l\in\mathcal{L}} p^t_l = 1$. The values p^t_l are computed from the training period using counting. That is, for each time of day t, we count the total number of users m^t_l recorded in region j at time t, and define $p^t_l = m^t_l / \sum_{l\in\mathcal{L}} m^t_l$. Hence, the prior defines an expected distribution over region counts for different times of the day.

The attacker receives noisy occupancy counts $(y^t_l)_{l\in\mathcal{L}}$ of all regions at timepoint t. Depending on the attacker type, certain regions in the buildings are opened. If the opening does not reveal the location of u immediately, it modifies the priors as $p^t_{ul} = \frac{p^t_{ul}}{\sum_{\ell\in\mathcal{L}_{closed}} p^t_{u\ell}}$ for $l \in \mathcal{L}_{closed}$, where $\mathcal{L}_{closed}$ is the set of regions that remained closed.

The posterior distribution is estimated using the results of Sec. 3.3.10.7 as

$$f_X(x|\mathbf{Y} = y) = \frac{\sum_{z=1}^{m} f_X(x|\mathbf{C}^g = z) \cdot f_Y(y|z) f_C(z)}{\sum_{z=1}^{m} f_Y(y|z) f_C(z)} \quad ,$$

where the probabilities are instantiated as follows.

- $f_X(x|\mathbf{C}^g = z) = \frac{z}{m}$.
- $f_C(z) = \binom{m}{z} p^z (1-p)^z$, where $p = f_X(x)$.
- $f_Y(y|z)$ depends on the particular analyzed privacy mechanism.

The quantity $f_Y(y|z)$ is approximated from (y, z) points of training data using *kernel density estimation* (KDE). We use Gaussian kernel of Python `scipy` library [73]. By default, the Gaussian KDE bandwidth parameter in `scipy` library is $n^{-1/(d+4)}$, where n the number of data points and d the number of dimensions. In our case, n comes from the training data (88 days), and since we are computing a separate kernel for each true count z (the approximated probability density is conditional), we have $d = 1$. We note that attacker's success may depend on the bandwidth parameter, and choosing one that approximates the noise distribution most precisely is out of scope of this work.

The posterior confidence of the attacker may be erroneous due to improperly computed noise distribution, or improperly computed priors. The latter may happen even if we use a well-defined DP mechanism. Hence, we add an important condition to our privacy metric. We model a particular attacker who actually *makes a particular guess* about victim's location. We then check whether that guess has been correct or not, and nullify the estimated advantage if the guess was incorrect.

Results. The results of our experiments are presented as plots (a sample plot is given in Figure 70). For each of the T timepoints, we count the total number of people whom the attacker managed to localize correctly with certain *confidence*, defined as the posterior probability of being in the room where the user has actually been according to the presence table. On each plot, the x-axis denotes the timepoint, and the y-axis is the number of localized people. The colors, ranging from light blue to dark red, correspond to localization confidence p, where light blue is the lowest confidence class ($0.0 < p \leq 0.1$), and dark red is the largest confidence class ($0.9 < p \leq 1$). Notice that the plot shows for each time point the total amount of people localized in the building broken into different confidence classes. This means that, for instance, out of the 123 people located in the building at 11:40am of the particular day in Figure 70, 22 are localized with the lowest confidence $0.0 < p \leq 0.1$, 55 with confidence $0.1 < p \leq 0.25$, 26 with confidence $0.25 < p \leq 0.5$, and 20 with confidence $0.5 < p \leq 0.75$. There are no red and dark red areas for 11:40am, so there have been no people localized with confidence $p > 0.75$. The plot format will be the same for all plots in this work, so we will avoid repeating labels and legends on further plots to conserve space.

We now compare different privacy techniques w.r.t. the attacker's success in breaking user privacy for the same level of utility.

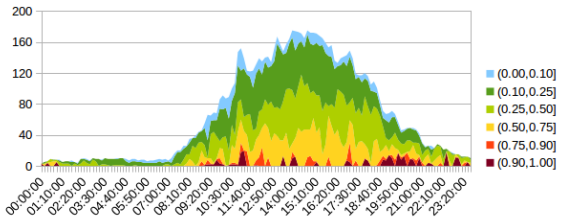


Figure 70: A Sample Plot: x-Axis is the Timepoint, y-Axis the Total Number of People Localized Within Each Confidence Class

Setting up Privacy Parameters. To fairly compare privacy loss across diverse PETs, their utility should be similar as there is an inherent privacy vs. utility tradeoff.

The *presence* database made available by the IoT CRT included aggregated data — the number of people present in each room of the Donald Bren Hall, recorded every five minutes. In fact, the same aggregations were available thrice — once as precise occupancy counts, once sampled as differentially private counts with $\epsilon = 1$, and once sampled as differentially private counts with $\epsilon = 0.1$. The last two datasets were computed with the help of the PeGaSus tool, representing one sample of its differentially private information release mechanism.

There was interest in the loss of utility that would occur, if one of the datasets with differential privacy were used instead of the precise one. To answer this question, one first has to specify the meaning of utility. It was discussed in the IoT CRT, and the utility was defined as "not missing significant events". Moreover, a **significant event** was defined as a room having an occupancy that is at least two standard deviations away from its average occupancy. To measure the utility, one thus has to determine the number of significant events in the dataset with precise counts, and then determine (i) how many of these events are missed in the differentially private dataset, and (ii) how many time an occupancy count in the differentially private dataset would count as a significant event, even though it is not significant in the precise dataset.

This approach makes "loss of utility" a property of a particular sample of the differentially private mechanism. A better description of the utility loss of the PeGaSus mechanism would consider this loss as a random variable (where the random choices are those made by the mechanism), and describe this variable by e.g. its mean, standard deviation, and other characteristics. Generalizing even more, the underlying presence dataset should also be varied, and a suitable summarization mechanism found. Still, despite the naïveté of this approach, it presents a clear way forward: it has clearly stated what "utility" means. This measure of the utility can now be targeted by the designers of differentially private information release mechanisms.

We have performed the necessary counts and determinations of significant events in the datasets that were made available. We made the following counts:

- The dataset with precise counts had 2,934,932 data points. Each data point states the occupancy count of a particular room (or a set of rooms) at a particular moment. The timepoints range from the beginning till the end of the month of February, 2018.
- The differentially private datasets had the same data points, but only for a subset of rooms. The number of data points in both datasets was 1,048,575. As all these points also existed in the precise dataset, we used these points to preform the utility analysis.
- Out of these 1,048,575 data points, 54,177 were determined to be "significant events". The other 994,398 data points did not represent significant events.
- The differentially private dataset for $\epsilon = 1$ contained 27,031 of these significant events, meaning that it missed 27,146 of them. Additionally, this dataset falsely characterized 36,979 non-significant events as significant. The remaining 957,419 data points were correctly characterized as non-significant.

- The differentially private dataset for $\epsilon = 0.1$ contained 14,444 of these significant events, meaning that it missed 39,733 of them. Additionally, this dataset falsely characterized 98,191 non-significant events as significant. The remaining 896,207 data points were correctly characterized as non-significant.

TIPPERS team has developed a tool which, given a specific task, obtains the configuration parameters per PET (i.e., epsilon and TTL) that will satisfy a given utility requirement. The parameters for PeGaSus, Laplace, and TTL that give us the utility 75% and 90% are summarized in Table 26. Since the privacy parameters for the same utility are very different for day and night time, different privacy parameters are extracted for day and night times. Since the utility is averaged over time and space, we also show average variance in utility over time and space σ_u.

Table 26: Parameters Achieving the Same Utility for Different Privacy Mechanisms

	utility	Laplace		PeGaSus		TTL	
		σ_u	ϵ	σ_u	ϵ	σ_u	T
day	75%	23.7	0.1	24.1	0.04	20.4	$1sec$
	90%	14.5	0.66	16.9	15	14.9	$2min$
night	75%	26.6	0.18	26.6	0.00001	19.5	$1sec$
	90%	19.3	1.675	19.5	4	19.5	$1sec$

Results per attacker type. We consider certain types of realistic attackers in the context of the dataset. There are 64 regions in the building, each having typically granting access to different profiles of people (e.g., students, professors, staff). We will consider three types of attackers based on such information.

1. An external attacker who is not present in the building and thus, does not have access to any region (i.e., 0 open regions).
2. A student (or a group of students) who coordinate an attack and gain access to all regions where a student can enter (i.e., 37 open regions). This includes classrooms and public areas.
3. A building administrator who has access to the security camera system and thus has access to all the regions covered by cameras (i.e., 39 open regions). This set of spaces includes public areas as well as corridors near offices.

Figures 71-73 compare different privacy techniques in different settings for the three types of attackers. The columns correspond to the four types of experiments (day/night, 75%/90%-utility), and the rows to different privacy mechanisms, including guessing from true counts (the last row). In each graph, the X-axis is time (7am-7pm for day and 7pm-7am for night in intervals of 10 minutes) and the Y-axis is the number of people correctly localized at each confidence level. Note that the scaling of Y-axis for day and night time are different, as the total number of people in the building is very different for them. As the noise added by DP techniques will be different in different executions, each experiment has been repeated n=30 times, and for each posterior probability class we took the average number of people that has been guessed with that probability. We consider that in the case of these realistic adversaries an open region implies that the adversary knows exactly who is inside of it. Thus, in the following we consider such information to be prior. In the plots we focus on how the different privacy mechanisms affect the guessability of the individuals in closed rooms.

For the weakest adversary (an external attacker), the practical privacy offered by Laplace and PeGaSus is almost the same than the one offered by TTL, which is much easier to set up. Even when the practical privacy for Laplace and PeGaSus is similar, for higher utility values the formal privacy guarantee for PeGaSus is less than Laplace during the daytime and nighttime (when ϵ gets extremely large to provide the same level of utility), whereas for lower utility values it is the opposite. When the adversaries become stronger (e.g., the student or administrator attacker), Laplace and PeGaSus offer more practical privacy than TTL, as it is expected. However, in some specific situations (e.g., in the

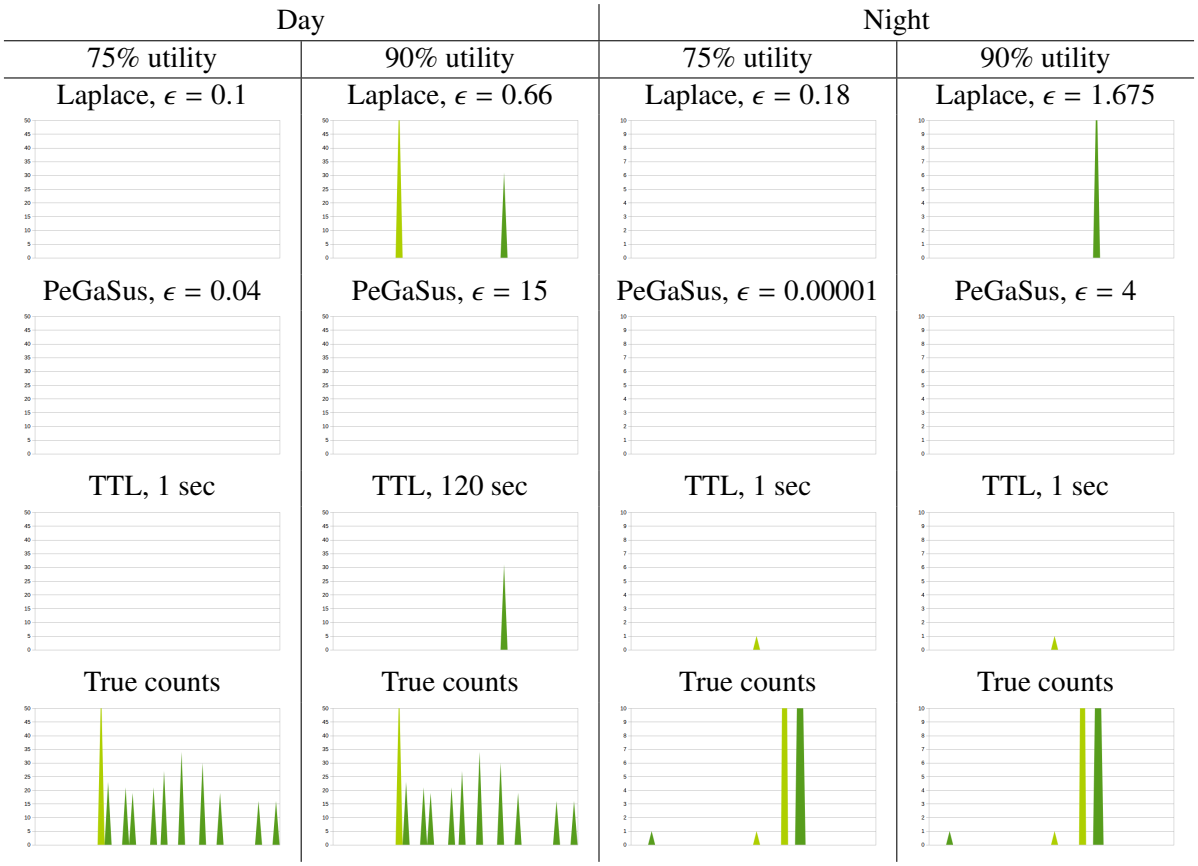


Figure 71: Comparison of Different Mechanisms for the External Visitor Attacker

afternoon when the building is less occupied) all the techniques behave similarly. Additionally, with stronger attackers the privacy loss due to the prior and adversarial knowledge at the time of the attack is high already. This means that effectively, in such situations the privacy of most of the individuals would be already compromised. Therefore, the difference between the differential privacy based techniques and TTL in terms of number of people being localized is small.

Experiments with theoretical attackers. We have also performed some more experiments with theoretical attackers with quantitatively defined strength. In particular, we assume that the attacker receives exact locations of some $\delta \in \{0\%, \dots, 90\%\}$ of the users in the system (the victim is not among these). This is done by opening some rooms at random, so that the number of people in those rooms is ca $\delta \cdot m$, where m is the total number of people recorded in the building. Opening the rooms modifies priors, as $p^t_{ij} = \frac{p^t_{ij}}{\sum_{k \in \mathcal{J}} p^t_{ik}}$ for $j \in \mathcal{J}$, where $\mathcal{J}$ is the set of rooms that remained closed. The attacker estimates posterior probability of the victim being in each room. Since the rooms are opened in such a way that the victim is not among these, the strongest attacker will be the one with largest δ.

The plots of prior probabilities are given in Figure 74. The three columns correspond to the initial knowledge of the attacker, where he already knows $\delta \in \{0\%, 50\%, 90\%\}$ of people locations. We see that, for larger δ, the attacker learns a lot already from priors. However, we will see that it gives larger advantage as well.

We demonstrate guessing advantage from known noise distribution on example of Laplace noise. We compute $P_Y(x, y, \mathsf{A})$ for a particular instance of noisy counts, $P_C(x, c, \mathsf{A})$ for a particular instance of true counts, and $P_P(x, \mathsf{A})$ for the general distribution of counts, assuming that we have the same number of timepoints, users, and rooms about which corresponding attackers make their guesses. The quantities $P_Y(x, y, \mathsf{A})$, $P_Y(x, y, \mathsf{A})$, $P_Y(x, y, \mathsf{A})$ are computed as in Sec. 3.3.10.7. The additional knowledge A depends on δ and just re-scales the priors as described above.

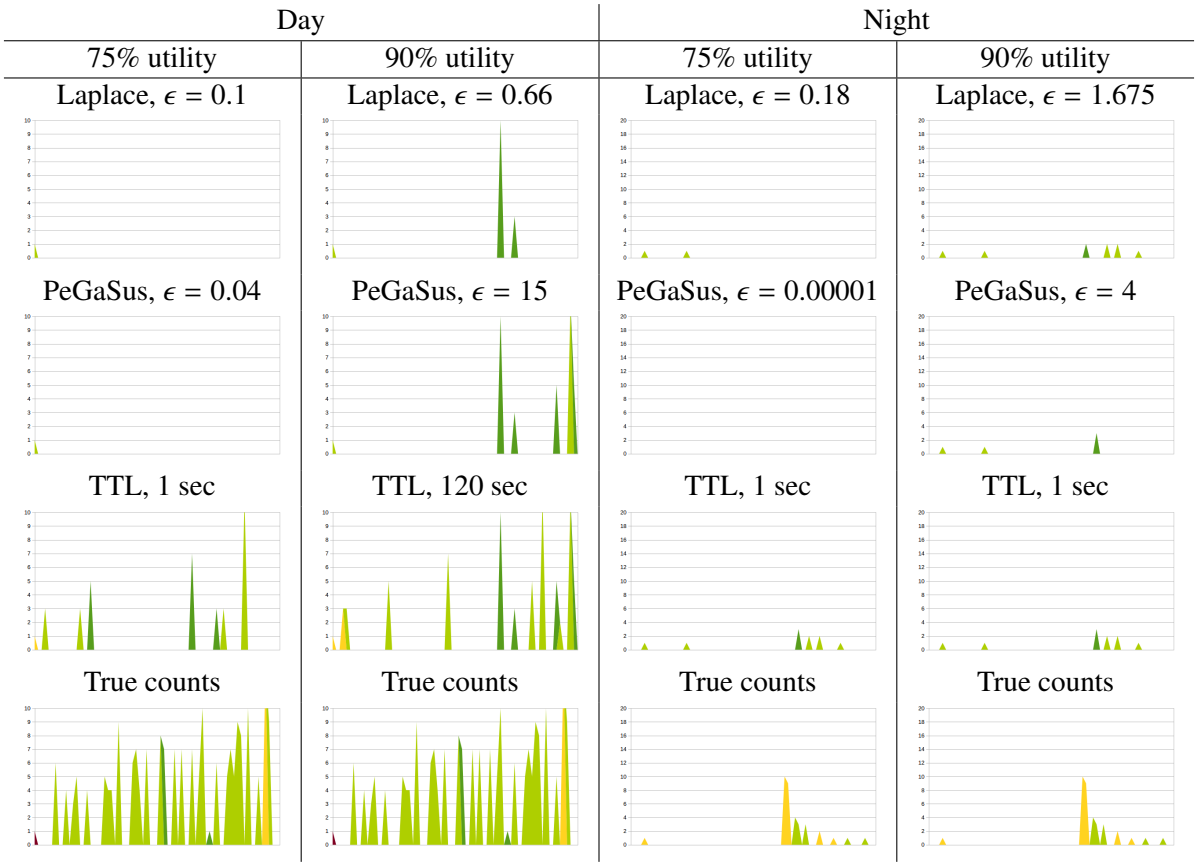


Figure 72: Comparison of Different Mechanisms for the Student Attacker

In Figure 75, we present results for $P_Y(x, y, \mathsf{A})$, where the rows of the plot matrices correspond to $\epsilon \in \{0.1, 1.0, 5.0, 10.0, \infty\}$, where ∞ is guessing from true outputs, and 0.0 guessing purely from prior, which we do not plot as it would give an empty graph with zero advantage. We can see how confidence increases with ϵ. It is interesting that $\epsilon \geq 5.0$ already gives us a plot very similar to guessing from true outputs, so it does not make sense to consider larger epsilons. For smaller epsilons, we indeed get smaller confidence, which converges to 0 as $\epsilon \to 0$. While the posterior *probability* always increases with δ, we see that the *advantage* may sometimes be larger for smaller δ, which means that the attacker guesses so poorly from prior that even a very noisy answer gives some benefits.

The posterior guesses are compared in Figure 76 for particular noisy outputs (left), particular true counts (middle), and for the general distribution (right). We have generated just one instance of noisy counts, and indeed with high probability we expect something similar that we would get in average for the particular true counts. The general distribution also gives us quite similar results. We see that the instance-based approach, which is the fastest to analyze (see Table 27 for time benchmarks), gives a reasonable estimate for the general distribution as well.

It is interesting to compare new results with the worst-case upper bound estimate of Sec 3.3.10.1, which does not depend neither on δ nor the particular counts, and holds for any ϵ-DP mechanism. The results are given in Figure 77 for $\epsilon \in \{0.1, 1.0, 5.0, 10.0\}$. We see that, for larger ϵ, the upper bound gets larger than the probabilities of guessing from true counts, so the upper bound is too rough for our type of attacker. Indeed, if we had the strongest DP attacker, then all guessing probabilities for true counts would be 1 in the worst case, as knowing everyone's location except the victim, together with counts, would leak the exact location of the victim. The bounds seem to be good for very small ϵ.

Table 27 shows the times of computing the posterior probabilities for $\delta \in \{0\%, 10\%, 25\%, 50\%, 75\%, 90\%\}$ (note that there were more δ instances tested, and we only plotted some of them). Since larger ϵ requires a finer partitioning of the integration over y, we report

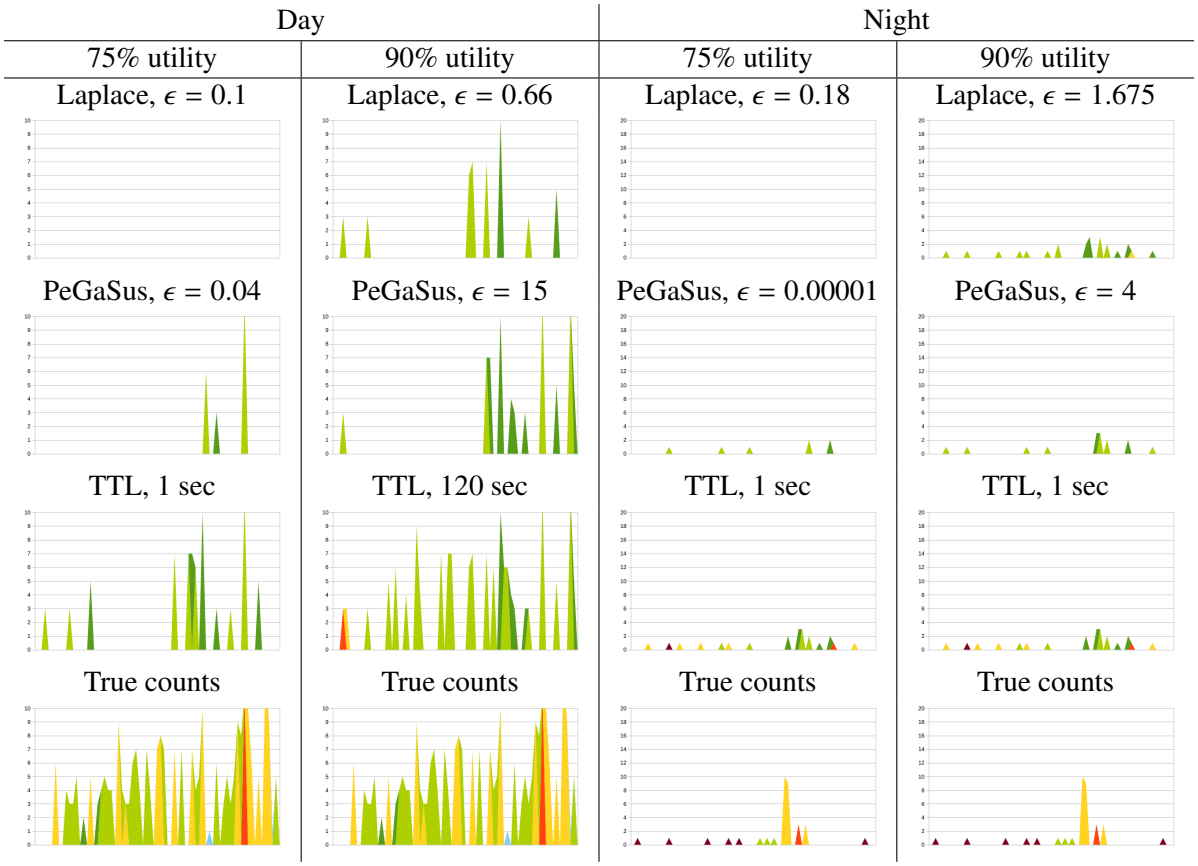


Figure 73: Comparison of Different Mechanisms for the Administrator Attacker

Table 27: Running Times (in Seconds) of Computing Posterior Probabilities for Laplace Noise

	Preprocess	$\epsilon = 0.1$	$\epsilon = 0.5$	$\epsilon = 1.0$	$\epsilon = 5.0$	$\epsilon = 10.0$	$\epsilon = 20.0$	total time
worst-case DP	4.8	0.58	0.57	0.57	0.62	0.60	0.62	8.8
$P_Y(x, y, \mathsf{A})$	5	5.1	5.1	5.2	5.4	5.5	5.5	38.4
$P_C(x, c, \mathsf{A})$	5.2	756	180	182	231	233	300	1898
$P_P(x, \mathsf{A})$	5.5	930	320	450	760	1020	1340	4840

different times for different ϵ values separately. We see that, while in general computation time grows with ϵ, this is also larger for $\epsilon = 0.1$ as well. The reason is that the noise of smaller ϵ has larger variance, so in the integration we need to cover a larger span of y values for which the noise is non-negligible.

Summary. Differential privacy in general consider very strong attackers that have access to almost unlimited information. In our set up with more realistic adversaries, we have seen that when the adver-

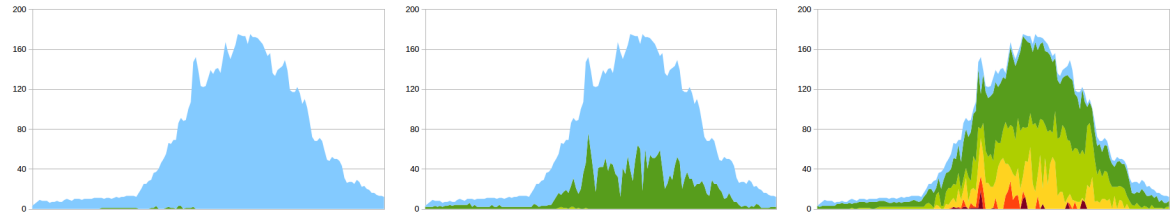


Figure 74: Prior Guessing Probabilities for $\delta = 0\%$ (Left), $\delta = 50\%$ (Middle), $\delta = 90\%$ (Right)

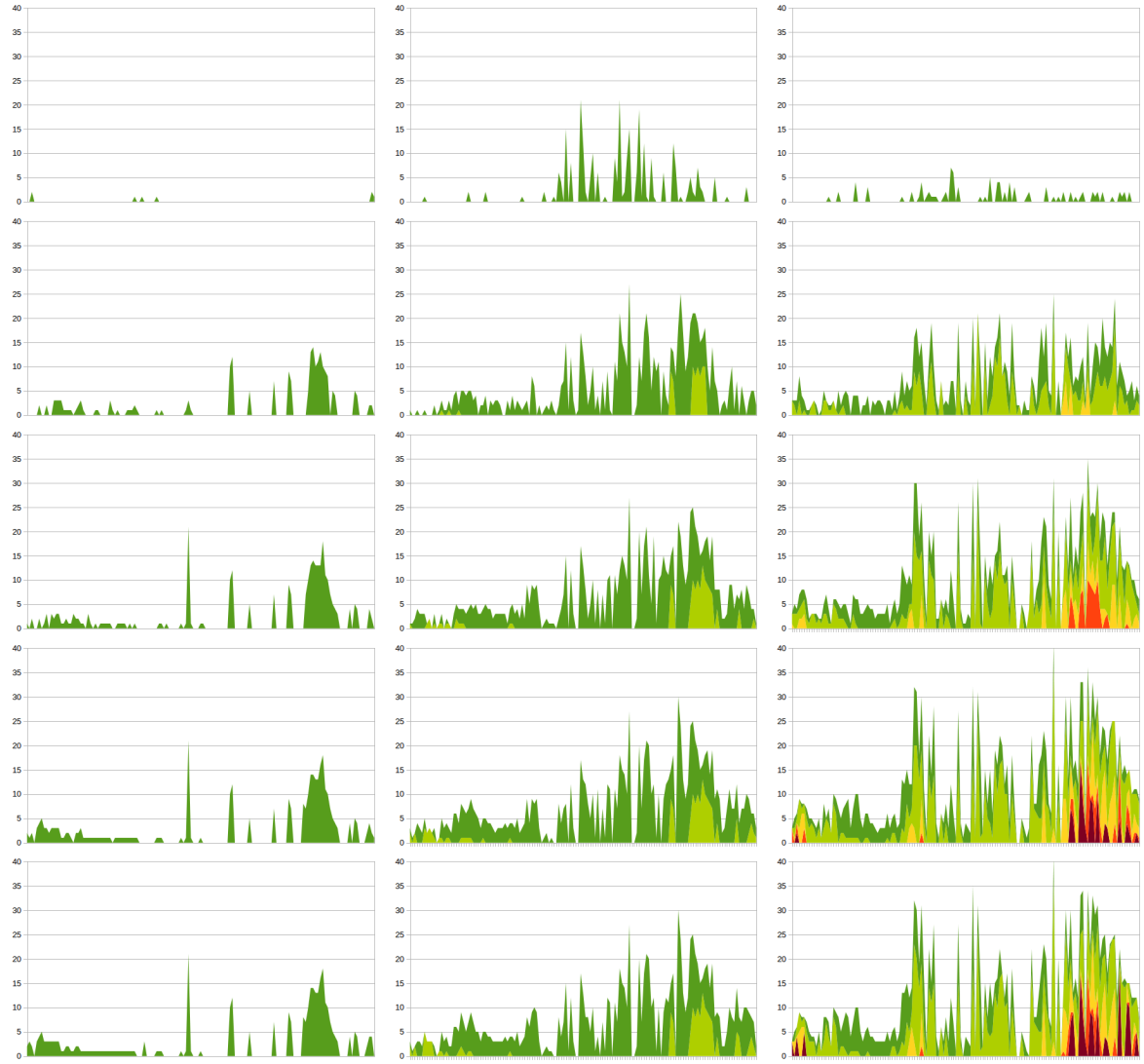


Figure 75: Posterior Guesses from Noisy Laplace Counts for $\delta = 0\%$ (Left), $\delta = 50\%$ (Middle), $\delta = 90\%$ (Right), $\epsilon \in \{0.1, 1.0, 5.0, 10.0, \infty\}$ (Top to Bottom)

sary is weaker (e.g., our external attacker). The practical privacy offered by Laplace and PeGaSus is almost the same than the one offered by TTL. Even when the practical privacy for Laplace and PeGaSus is similar, for higher utility values the formal privacy guarantee for PeGaSus is less than Laplace during the daytime and nighttime, whereas for lower utility values it is the opposite. When the adversaries become stronger (e.g., the student or administrator attacker), Laplace and PeGaSus offer more practical privacy than TTL, as it is expected. However, in some specific situations (e.g., in the afternoon when the building is less occupied) all the techniques behave similarly. Additionally, with stronger attackers the privacy loss due to the prior and adversarial knowledge at the time of the attack is high already. This means that effectively, in such situations the privacy of most of the individuals would be already compromised. Therefore, the difference between the differential privacy based techniques and TTL in terms of number of people being localized is small.

We would like to highlight that even when in terms of practical privacy the techniques behave similarly, TTL lacks of formal privacy guarantees which means that stronger attackers using more sophisticated attack methods could potentially result in higher privacy loss. Additionally, the results for Laplace and PeGaSus, which are the average over 30 counts, could potentially be worse depending on the noisy count generated in a single run at publishing time. Comparing the distributions of attacker's success for different privacy mechanisms remains out of scope of this work.

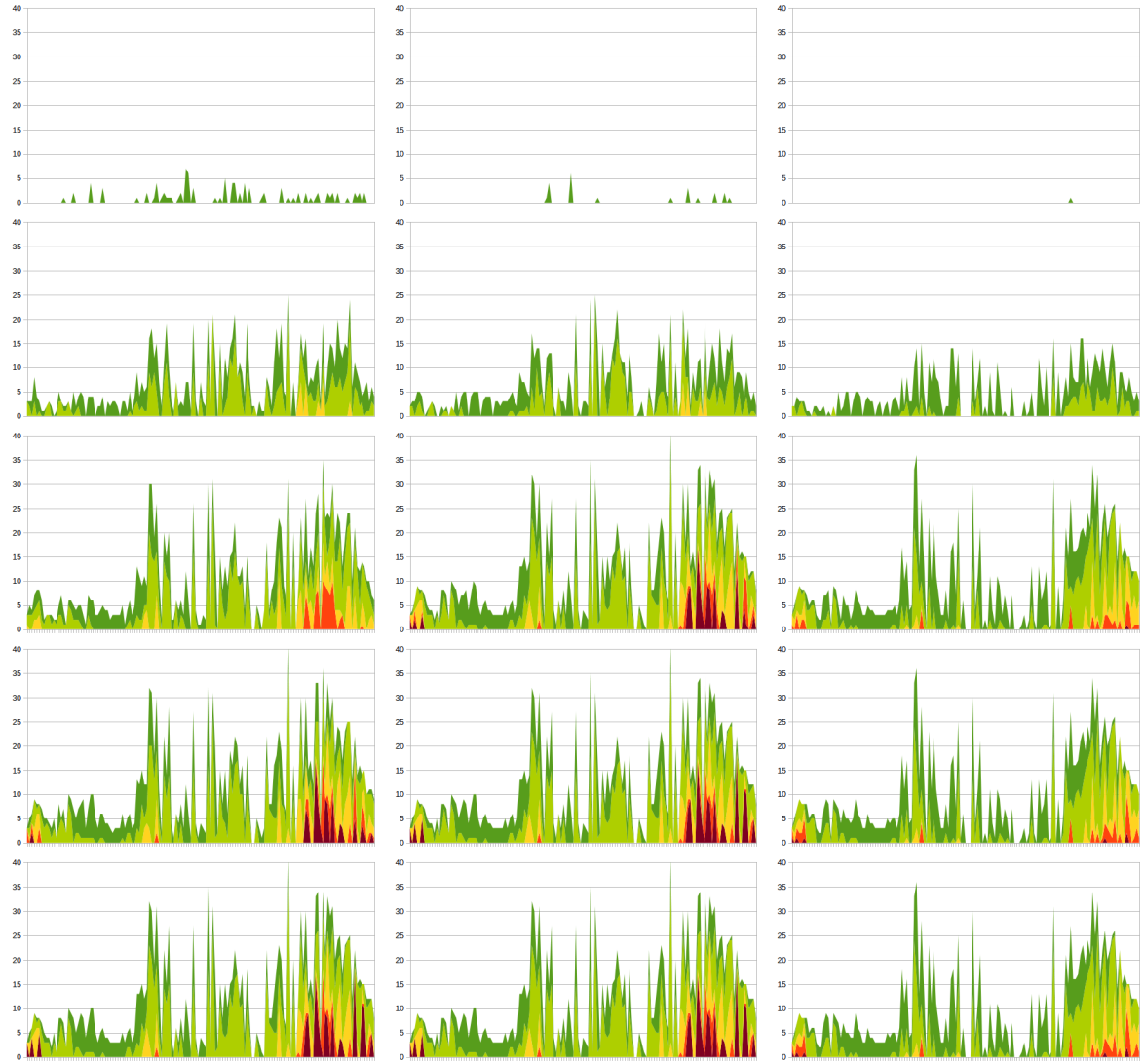


Figure 76: Posterior Guesses for $\delta = 90\%$ from Noisy Laplace Counts (Left), True Counts (Middle), and Count Distribution (Right), $\epsilon \in \{0.1, 1.0, 5.0, 10.0, \infty\}$ (Top to Bottom)

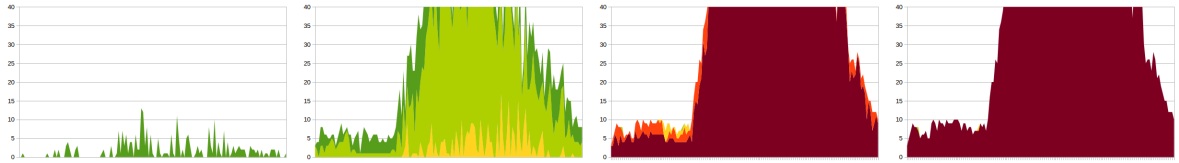


Figure 77: Posterior Guesses for Worst-Case DP, $\delta = 90\%$, $\epsilon \in \{0.1, 1.0, 5.0, 10.0\}$ (Left to Right)

4.5.3.2 Trident Warrior Scenario. In Trident Warrior scenario, TIPPERS system is used to collect location data of sailors on a ship. This data can be used to further compute various aggregated statistics (e.g., occupancy of spaces along time, average time spent by an individual in a particular location, average number of people an individual interacted with, etc.). Since the underlying data is private (i.e., location of individuals along time), the idea of an aggregated statistics leak of information about individual records is a precautionary concern. Well-known mechanisms can be applied like Differential Privacy (DP) to protect individual records, but knowledge on how to choose the appropriate privacy parameters is required. In this section, the different values of ϵ for a DP mechanism are estimated.

Assume a data table is displayed with user-location-time-table consisting of categories such as: `user`, `day`, `location`, `daytime`, and `time_spent`. The categories are used to describe the amount of time the user has spent on each area per day. Let us consider, for instance, a commanding officer that observes aggregated averages on the time spent for each recorded location+daytime combination, stated as the following query:

```
SELECT day, location, daytime, AVG(time_spent)
FROM user-location-time-table
GROUP BY day, location, daytime;
```

The goal is to estimate how much a commander (treated here as an adversary), can learn about the particular time spent by a specific user. A quite strong attacker that may already have knowledge regarding the exact amount of time spent by other people who have been together with the victim at the same time in the same location can be assumed. This idea is motivated by the definition of differential privacy, which is aimed to protect against such attackers.

Experiment. In our experiment, the attacker first fixes a single victim out of n users. It computes the prior assumption of the victim's data based on the data of the other n-1 users (or only a certain fraction of these users). The attacker then tries to guess the victim's spent times based on the prior it has already learned, and on the aggregated statistics that depend on the victim's data. We assume that the attacker wins even it does not guess the spent time precisely, but with some precision. For example, if the attacker says that a user has been in a room for 17 minutes, but it actually was 17.5 minutes, the guess is still considered sufficiently correct.

There are n users that participate in the experiment. For each location+daytime+day combination, each user u_i has spent times distributed according to normal distribution $N(\mu_i, \sigma_i)$. The attacker predicts μ_i and σ_i based on the data of the other $n-1$ users.

Fixing some posterior probability t, (e.g., $t = 0.9$), we want to compute the precision r within which the attacker's guess stays with probability t. For example, if the actual time is x_0, then with probability t the attacker's guess will be $x \in [x_0 - r, x_0 + r]$.

It can be assumed that an ϵ-DP mechanism is applied to the released average. In particular, the sensitivity of AVG query w.r.t. attribute `time_spent` is $1/n$, so e.g., Laplace mechanism $Lap(\lambda)$ where $\lambda = 1/(n \cdot \epsilon)$ can be used. We note that our results show which ϵ would be sufficient for *any* DP mechanism. Using results from Sec. 3.3.10.1, if p_i is the prior guessing probability, then the posterior p_i' is bounded by

$$p_i' \leq \frac{1}{1 + e^{-a_i\epsilon}(1 - p_i)/p_i} \ , \tag{46}$$

where a_i is in this case the largest possible time that the user u_i might have ever spent in the analyzed location. Since normal distribution is unbounded, we formally have $a_i = \infty$, which makes the upper bound trivial. Given that the average is taken over a certain time span (e.g. an hour), we could take a_i equal to the length of that time span. However, we can do better. For a normal distribution $N(\mu_i, \sigma_i)$, we have $\Pr[x - \mu_i \leq a_i] = \mathsf{erf}\left(\frac{a_i}{\sqrt{2}\cdot\sigma_i}\right)$ for any $a_i \in \mathbb{R}$, where erf is the error function. If we take e.g., $a_i = 3\sqrt{2} \cdot \sigma_i$, we get $\Pr[x - \mu_i \leq a_i] = \mathsf{erf}(3) \approx 0.9998$, which essentially covers the set of possible inputs. A smaller value of a_i can be taken to reduce the size of the exponent, but it also reduces the

attacker's search space, so this parameter can be optimized to improve the upper bound on guessing probability. In our experiments, we have started from $a = 2 \cdot \sqrt{2} > 3$, computing $a_i = a \cdot \sqrt{2} \cdot \sigma_i$, and reducing a by 0.01 until $a < \frac{r}{\sqrt{2} \cdot \sigma_i}$, i.e. until we start getting $a_i < r$ and hence $p'_i < p_i$. In the end, we take a_i that results in the smallest p'_i.

For a fixed guessing precision r the prior probability is computed as

$$p_i = \Pr[x \leq x_0 - r] - \Pr[x \leq x_0 + r] = \frac{1}{2} \mathsf{erf}\left(\frac{|x_0 + r - \mu|}{\sigma \sqrt{2}} \right) ,$$

and the posterior p'_i is computed from p_i and ϵ as in Eq. 46 with $a_i = a \cdot \sqrt{2} \cdot \sigma_i$ for $a \in [r/(\sqrt{2} \cdot \sigma_i), 2 \cdot \sqrt{2}]$.

We perform experiments for $r \in \{0.5, 1.0, 2.0, 3.0, 4.0, 5.0\}$ and $\epsilon \in \{0.1, 0.5, 0.75, 1.0, 2.0, 3.0, 4.0\}$. For each ϵ, we find the smallest r for which $p'_i \geq t$.

Results. Since the actual data of the users' movement on the ship collected during the exercise is classified and cannot be shared even for the privacy study, the researchers of TIPPERS team simulated the behavior of 150 users, such that the replicated data has the same statistical moments (means and standard deviations) as the actual data. This imitated data serves as the "real" data for the privacy study. Therefore, ϵ-differential privacy is applied to this dataset and the adversary's success is computed.

The posterior guessing probabilities for 5 data samples have been estimated. The results are depicted in Figure 78. The number of spent times guessed is plotted with probability ≥ 0.9 for different precisions, where the precisions are represented with different colors. The dark green color represents the roughest guess (±5 minutes), and the dark red color the most precise guess (±0.5 minutes). For $\epsilon = 0.1$, only few people are depicted in the bar, and this means that for the others the guessing precision was more than ±5 minutes.

While the datasets are different, similar trends in these five plots are noticed. If $\epsilon \geq 3$ is taken, then very little privacy guarantees are obtained, and each user's spent time may be guessed within one minute of precision. On the other hand, for $\epsilon \leq 0.5$, the guessing precision ranges between 4 and 5 minutes, which is much better, considering that the actual spent times are on average 8-9 minutes in the given datasets. There are always several people for whom the guessing probability is large even for small ϵ, as their behavior is more predictable, but there are not too many such people.

Summary. We have performed experiments to check out which values of ϵ would be sufficient to give certain privacy guarantees under certain assumptions. Since smaller ϵ means more noise in aggregated statistics, data utility also needs to be taken into account, which would be a separate study and depends on how the statistics are actually going to be applied.

Alternatively, we could study weaker attackers who know time durations of only some of the other users. In that case, it could be possible to get better privacy for larger values of ϵ. Modeling a particular attacker would require knowledge about the context, who the attacker is and what he already knows. This remains out of scope of this privacy study.

So far, we have estimated the severity of guessing precision equally for all users. E.g. we have considered guessing precision $\pm 5 min$ safe enough not to consider more coarse predictions. These numbers have been taken based on the actual (simulated) presence data, where the distributions of different users have been quite similar to each other. In some other experiments, we would need to consider different guessing radiuses, if the times are longer or shorter. We propose that a more generic privacy metric would be to consider a *different* guessing radius r_i for each user, making it dependent on the standard deviation σ_i, expressing r_i as a certain quantile of the distribution.

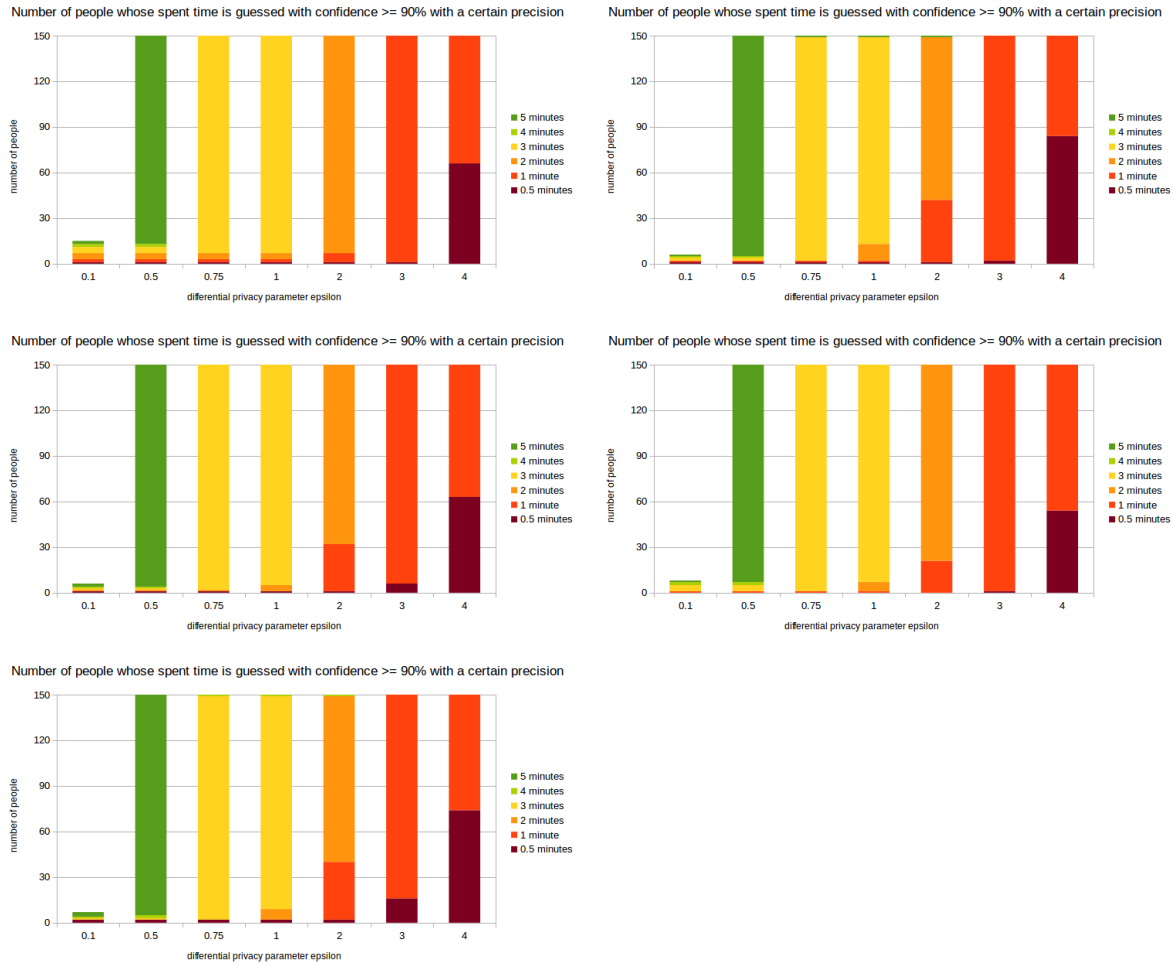


Figure 78: Precisions of Guessing a Particular Spent Time with Confidence $t = 0.9$ for Different ϵ Values and 5 Different Datasets

4.6 Secure Multiparty Computation in NAPLES

4.6.1 Deliverable Outcomes. The main deliverable outcomes of the work carried out under this subtask are:

- Solution analysis and design document - provides an overview of any analysis carried out (including use case specification) and details the prototype implementation specifics.
- Prototype implementation, including:
  - Final easily testable product containing the respective Docker image. When run according to the usage instructions the respective Docker container should start generation of the synthetic log entries, runs the non-privacy preserving version of the algorithm, then runs the privacy-preserving version of the algorithm in SecreC using the Sharemind emulator, and then compares the results.
  - Source code of prototype implementation.
  - User and install guides for the prototype are included in code delivery as files INSTALL.md and USAGE.md.
- Final reporting (this document) to provide an overview of work carried out.

Handover of these results to the Cyber PA project will be discussed during Q1 of 2021.

4.6.2 Analysis and Prototyping Main Outcomes. An important outcome from prototyping (in addition to a working prototype and associated learnings) is the proposed event correlation approach / algorithm. In principle, the prototype event correlation algorithm attempts to construct chains of (correlated) log events by iterating over the input log entry values in timestamp order. If the value, iterated over, already matches a value of the last log entry in an existing chain, the log entry will be appended to the existing chain. Otherwise, a new chain is created. As output from the algorithm, each data owner receives log entries from detected chains which contain at least one log entry from the respective data owner. Chains with only one log entry are not considered for output.

In addition to using Pleak to model processes within the Cyber PA project, the event log correlation process was also modelled in Pleak. The model in Figure 79 visualizes the stakeholder involved in the exercise, their activities and the data elements that are used and / or generated during the processing. In addition, the scheme specifies security methods for data flows and an overview of the visibility of data elements to the stakeholders is provided. Finally, Leakage detection analysis was run on the model to give a more thorough overview of the process security guarantees).

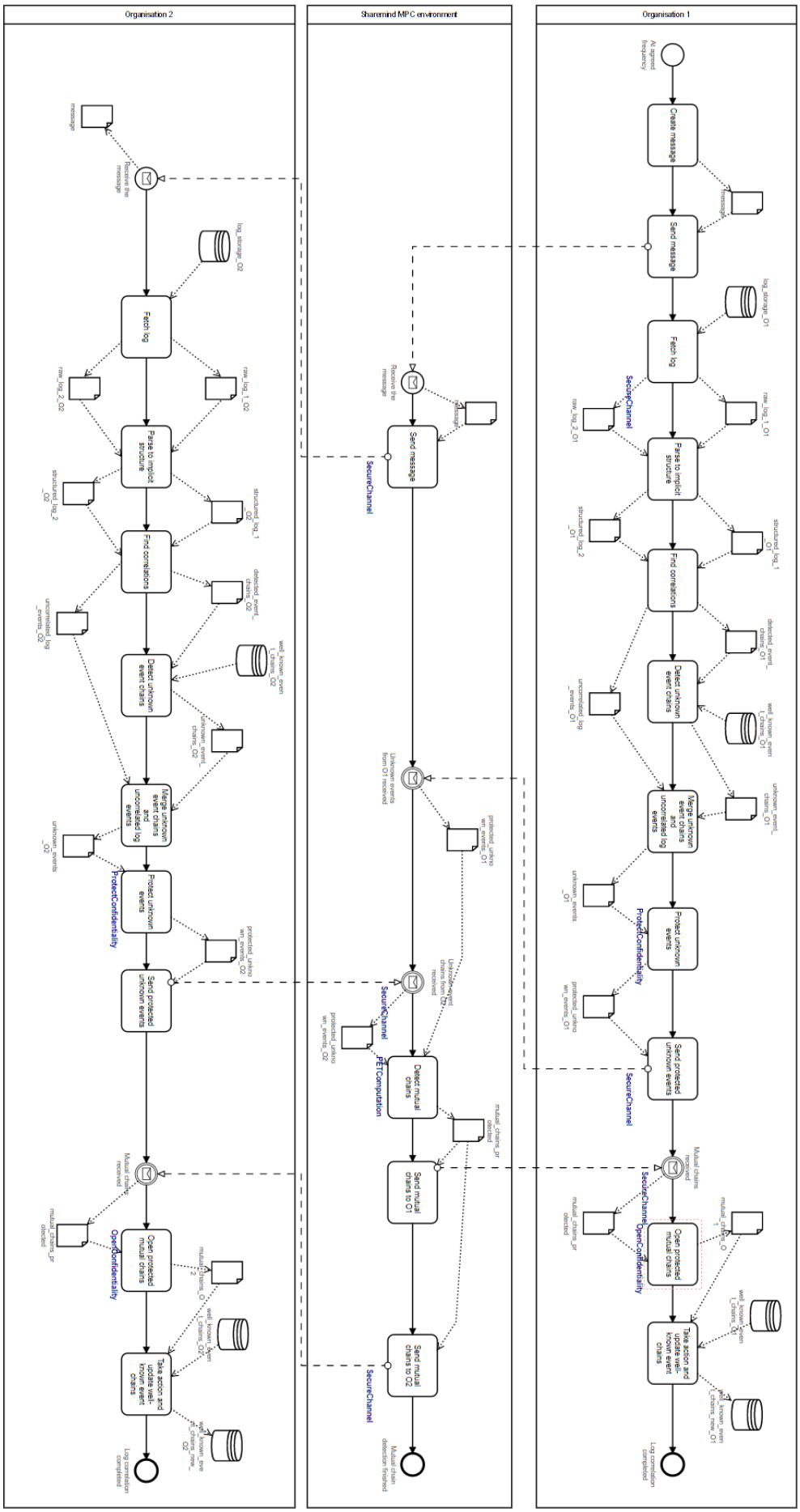


Figure 79: Event Log Correlation Process in PE-BPMN Notation

4.6.2.1 Stakeholders. Stakeholders are individuals, groups, organizations or machines (hardware and software) without whose support the project would cease to exist. This section describes only primary stakeholders directly involved in the exercise. Although, in real project, secondary stakeholders are very important and should be elicited.

Table 28: Event Log Correlation: Overview of Stakeholders

Name	Description
Organization 1 & 2 (ORG1/ORG2 or ORG1/2)	Represent organizations interested in participation of system log contribution and wishing to learn from mutual event log correlation exercise. In the current model there are only two organizations (1 & 2) involved, but the number of participating organizations is not limited and the exercise can be conducted with as many organizations as needed.
Sharemind MPC environment (MPC)	We are using Sharemind MPC environment as a secure computation environment which requires at least 3 non-colluding hosts in order to provide the required security guarantees. This stakeholder consists of 3 separate organizations that are chosen by Organization 1 & 2 or $1 \ldots n$ to host the Sharemind MPC instances. It might be that participating organizations themselves are the hosts, but they can also be external participants as long as Organizations are convinced that there is very low likelihood for these hosts to collude.

4.6.2.2 Activities. Table 29 lists the activities of the process. For every activity name and description is used to give an overview what is done during the activity. Additionally, what triggers the activity (pre-conditions) and what results after the activity (goal) is described in separate fields. Finally, a tool (computer application most of the time) that is used to perform the activity is also mentioned.

Table 29: Event Log Correlation: Overview of Activities

Name	Actor	Description	Trigger	Goal	Tool
Create message	ORG1	On the current model ORG1 is an initiator who informs all the participants about the beginning of the exercise by creating a message.	On agreed time.	Create informative message.	Can be an email or any other common communication channel.
Send message	ORG1 & MPC	ORG1 sends the message, created in the previous task to MPC participants. MPC participants forwards the message to other contributors when the environment is set up. NOTE: The informing part of the process is somewhat irrelevant, but is required for the Pleak tool used to analyze the process.	If ORG1: message created. If MPC: messages from initiator received and all MPC participants confirmed readiness.	Inform ORGs and MPC about the beginning of exercise.	Can be an email or any other common communication channel.
Fetch log	ORG1/2	Organizations choose systems which logs they wish to use in the exercise and gather them for processing.	Start message arrived, but can be done in advance to the exercise.	Logs gathered.	Dependent on the systems the ORGs are using.

Continued on next page

Continues from previous page

Name	Actor	Description	Trigger	Goal	Tool
Parse to implicit structure	ORG1/2	As stated, most event logs are text messages intended for humans to read. During this activity logs are been parsed using natural language processing parsing methods.	Logs gathered.	Logs converted to implicit structure and saved.	Regular expressions can be used or extracted manually by writing respective grammar or automatically using some pattern recognition algorithm.
Find correlations	ORG1/2	Organization finds correlations between different log entries. For this task we define correlation as a fact that two parsed log entries have at least one pair of equal variables. We define a log entry set as an ordered subset where log entry time stamps are strictly growing and sequential log entry pairs are correlated. Log entry set should then indicate presence of causally connected sub-events.	Event log entries are parsed to structured form.	Event chains detected.	Part of event correlation algorithm logic.
Detect unknown event chains	ORG1/2	Detected event chains are compared to already known event chains. In case the event chain is known it is no interest to the exercise. In case the event chain is unknown we record them for the exercise.	Event chains detected.	Unknown event chains separated from well-known event chains.	Part of event correlation algorithm logic.
Merge unknown event chains and uncorrelated log events	ORG1/2	During the log correlation activity some of the events that were not recognized as part of any chain and the events in the unknown event chains are merged into mutual data file for joint correlation exercise. Also, the transfer to the pre-agreed format is done during the activity.	Unknown event chains separated from well-known event chains and also events without chain connection detected.	File with pre-agreed structure of unknown events created.	Part of event correlation algorithm logic.

Continued on next page

Continues from previous page

Name	Actor	Description	Trigger	Goal	Tool
Protect unknown events	ORG1/2	Unknown events are secret shared for MPC environment.	File with pre-agreed structure of unknown events created.	Protected unknown event file(s) created. NOTE: Protected here means the data file is split using secret sharing technique into 3 shares where none of the shares contain any meaningful information separately.	Sharemind CSV Importer data upload tool.
Send protected unknown events	ORG1/2	The shares of protected unknown event files are uploaded to the MPC instances.	Protected unknown event file(s) created.	Protected unknown event file uploaded to MPC environment.	Sharemind CSV Importer data upload tool.
Detect mutual chains	MPC	The unknown events of all participating organizations are gathered and similar correlation activities described in Find Correlations activity are used to detect mutual chain events.	Protected unknown event file uploaded to MPC environment.	Mutual chain events dataset created.	Sharemind MPC servers running pre-agreed SecreC code.
Send mutual chains to O1/O2	MPC	Extract from the mutual chain events dataset chains containing events contributed by ORG1/2 and make them available to ORG1/2. NOTE: In case more than 2 organizations participating in the exercise only the chain events containing events contributed by the organization are made available.	Mutual chain events dataset created.	Mutual chain events published to the ORG1/2.	Custom Sharemind MPC client application.

Continued on next page

Continues from previous page

Name	Actor	Description	Trigger	Goal	Tool
Open protected mutual chains	ORG1/2	Open the mutual chain event records for processing. NOTE: Send and Open mutual chains to O1/O2 events in real life can be considered as one as same client application is most likely used for making the query. The method is merely used to fully benefit the Pleak tool.	Mutual chain events published to the ORG1/2.	Mutual chain events available for further processing for ORG1/2.	Custom Sharemind MPC client application.
Take action and update well-known event chains	ORG1/2	Take appropriate action based on what is learned from the exercise and update the known database accordingly	Mutual chain events available for further processing for ORG1/2.	Required actions taken and well-known event chains database updated.	Tools dependent on organization internal tools.

4.6.2.3 Data Elements. Data elements are information or data artefacts created or used during the process. They are either needed as inputs for activities or output of activities. In table 30 you find Name of the element, role that created and holds the element (Owner) and short description. Same data element can be used in multiple places, data element created during one activity is very often an input for the other.

Table 30: Event Log Correlation: Overview of Data Elements

Name	Owner	Description
message	ORG1	Simple text message for informing the beginning of the exercise.
log_storage_O1/O2	ORG1/2	Log storage of the system of an organization that the organization wished to use in the exercise.
raw_log_1_O1/O2, raw_log_2_O1/O2	ORG1/2	Log file in its initial form after extracting it form systems log storage.
structured_log_1_O1/O2, structured_log_2_O1/O2	ORG1/2	Log file parsed to the implicit structure for the log correlation activity. Each organization might have different structure and can convert the file to the mutual structure later, but also it can be already using the agreed structure.
detected_event_chains _O1/O2	ORG1/2	Result of correlation detection activity where event entries are marked with common identifier that represents membership to a chain. The events are ordered sequentially by time of appearance.
uncorrelated_log_events _O1/O2	ORG1/2	List of structured log events not belonging to any chain. It is structured the same way as structured_log_1_O1/O2
well_known_event_chains _O1/O2	ORG1/2	A database of known event chains of the organization. Structured the same way as detected_event_chains_O1/O2 so that it would be easy to query whether the detected chains match to know chains. Database is updated regularly by the organization once it detects new event chains. Also, the update takes place during the exercise.

Continued on next page

Continues from previous page

Name	Owner	Description
unknown_event_chains _O1/O2	ORG1/2	List of detected chain events that did not match to any of the known events recorded in the well_known_event_chains_O1/O2 database. Structured same way as detected_event_chains_O1/O2.
unknown_events_O1/O2	ORG1/2	List of events that could not be assigned to any chain during the correlation process. Structure is pre-agreed by stakeholders and is most likely a CSV file as this is the form required by Sharemind CSV Importer.
protected_unknown _events_O1/O2	ORG1/2	A secret-shared copy of unknown_events_O1/O2. Sharemind CSV Importer tool secret shares the unknown_events file into 3 shares where none of the shares contain any meaningful information separately but is processable as normal unknown_events file in Sharemind MPC environment. NB! In case MPC instances collude, the information is likely extractable from the shares.
mutual_chains_protected	MPC	The result of Sharemind MPC mutual chain detection activity where events of all the contributing organizations are correlated into mutual chain events. It is structured in the pre-agreed format so that the organizations can clearly understand and process the information. Still the data is in the protected (i.e., secret-shared) form and each share is controlled separately by one of the non-colluding MPC parties.
mutual_chains_O1/O2	ORG1/2	Subset of mutual_chains_protected, where only chains containing events contributed by the organization are present. Also, the data is now in the open form and fully readable and processable to the organization. To emphasize the organization sees only chains that contains events it contributed. In case there are more than 2 organizations involved in the exercise the events not containing any events contributed by organization X are not included. The data is structured the same way as mutual_chains_protected.
well_known_event_chains _new_O1/O2	ORG1/2	The same data element as well_known_event_chains_O1/O2 except containing updates of findings of the exercise.

4.6.2.4 Security Measures. Security methods used in the process, highlighted in blue in the process model:

- **SecureChannel** - Data transmission marked with SecureChannel in the PE-BPMN figure means that an authenticated and encrypted security channel is in use. Technically, we expect the use of standard solutions (e.g. TLS, Transport Layer Security) with up-to-date technical solutions. The communication channel set up in this way does not leak data to the communication service provider or other parties involved in the data transmission.
- **PETComputation & ProtectConfidentiality** - Data protection mechanisms based on the capabilities of the Sharemind MPC technology.
- **OpenConfidentiality** - OpenConfidentiality is an opposite action to ProtectConfidentiality and is also a data protection mechanism based on the capabilities of the Sharemind MPC technology.

4.6.2.5 Disclosure & Leaks-When Analysis. Using Pleak tools & capabilities on the process model Fig. 79 allows to generate reports that provide an overview of data elements by stakeholders and help detect / ensure data is secured.

The Leaks-When analysis result in Figure 80 shows that the unknown events are computed by the organizations on their own and are only depending on their own input data. However, the new chains are the ones computed together and are also affected by the logs of the other party.

	log_storage_O1.log	log_storage_O2.log	well_known_event_chains_O1.chains	well_known_event_chains_O2.chains
unknown_events_O1.data	if	never	if	never
unknown_events_O2.data	never	if	never	if
well_known_event_chains_new_O1.chains	if	if	always	if
well_known_event_chains_new_O2.chains	if	if	if	always

Figure 80: Leaks-When Analysis Result for the Prototype Model

The disclosure analysis result in Table 31 provides an overview of the visibility of data elements by stakeholder. Simple disclosure report is a table which lists all stakeholders and shows, whether and how the stakeholder sees a data element. Each cell is marked either V (visible), H (hidden) or –. Marking "–" means that the stakeholder does not see the data element in any way in the process. V means that contents of the data element are fully visible. H means that the stakeholder has the data element but this element is protected with security measures. For example, the Sharemind MPC environment has the data element containing information of mutual chains, but as it is protected, the content is not visible. In addition, cell can be marked with O, meaning that the stakeholder is the creator/owner of the data element and it is fully visible to the stakeholder. The table helps to understand what information is available to which parties.

In general, the reports confirm that neither the events nor the event chains of the contributing organizations are revealed to any other stakeholder and thus no confidential data is leaked to any unwanted participants. The Sharemind MPC hosts only see data in hidden form as shares of secret sharing.

4.6.3 Main Results. Joint situational awareness is built up from national situational awareness which in turn is built up from information gathered from local institutions and private enterprises. Local situational awareness is the result of the analytical work of the cyber security professionals. Many or even most private enterprises do not have cyber security specialists available. One way for such enterprises to participate in creation of the situational awareness is to share system logs from their computer and networking equipment with institutions with better analyzing capabilities.

As per analysis it was proposed to create an IT system log analysing engine which enables sharing potentially sensitive information in a privacy-preserving way, so that

- participating organizations would have a clean, easy, meaningful way to decide which information to share.
- participants would get strong guarantees on how the information is processed and what part in what aggregation level is published to other participating parties.

Table 31: Event Log Correlation: Simple Disclosure Report

Data element	ORG 1	ORG 2	MPC	Shared Over
detected_event_chains_O1	V	-	-	-
detected_event_chains_O2	-	V	-	-
log_storage_O1	O	-	-	-
log_storage_O2	-	O	-	-
message	V	V	V	S
mutual_chains_O1, mutual_chains_O2, mutual_chains_protected	V	V	H	S
protected_unknown_events_O1, unknown_events_O1	V	-	H	S
protected_unknown_events_O2, unknown_events_O2	-	V	H	S
raw_log_1_O1	V	-	-	-
raw_log_1_O2	-	V	-	-
raw_log_2_O1	V	-	-	-
raw_log_2_O2	-	V	-	-
structured_log_1_O1	V	-	-	-
structured_log_1_O2	-	V	-	-
structured_log_2_O1	V	-	-	-
structured_log_2_O2	-	V	-	-
uncorrelated_log_events_O1	V	-	-	-
uncorrelated_log_events_O2	-	V	-	-
unknown_event_chains_O1	V	-	-	-
unknown_event_chains_O2	-	V	-	-
well_known_event_chains_new_O1	V	-	-	-
well_known_event_chains_new_O2	-	V	-	-
well_known_event_chains_O1	O	-	-	-
well_known_event_chains_O2	-	O	-	-

V = visible, H = hidden, O = owner, S = SecureChannel

A system having the above properties was analyzed, modelled and prototyped as a part of the work task. Main findings from the prototyping activity were:

- The approaches taken give organizations ability to make clear distinction between mundane, possibly confidential every-day events and unusual, possibly harmful events which should be analyzed further. This methodology should be useful even without following information sharing and joint analysis. Therefore, it should increase the will and likelihood for enterprises to join the Cyber PA project information gathering network.
- Guarantees on how shared information is processed and revealed are provided by
  - implementing the correlation algorithm in Sharemind MPC.
  - carrying out Pleak analysis on modelled processes (Data Leakage analysis)
- Prototype benchmarks show overall asymptotic complexity $O(n^3)$ for chains detections algorithm, where n is the number of input system log records, whilst the non privacy-preserving version of chains detection has a complexity of $O(n \log n)$. This because the privacy preserving version is much more complex as it hides array access patterns, which when observed, might leak content or content distribution of the array.

For the purposes of the Cyber PA project the following prototype enhancement possibilities should be discussed, analyzed and tested:

- Improve the algorithm in a way that it does not need to look up things from arrays, thus not adding asymptotic complexity.
- Give up some insignificant amount of the privacy to significantly reduce complexity. This trade-off should be well thought-through and documented.
- Improve Sharemind MPC so required calculations do not have so much overhead.
- Consider some other privacy preserving technologies wish less computational overhead, taking into account the requirements which led to preliminary implementation on Sharemind MPC.

In conclusion, results from this task are a valuable additions to the Cyber PA project. Modelling carried out in Pleak helped to further understand the designed processes in the start of prototype implementation and further helped verify privacy guarantees of the Solution. Actual prototyping of the solution provided useful feedback on the usability of the prototype in a potential Cyber PA Solution framework. All this provides an improved starting point of enabling such functionalities as a part of the Cyber PA in order to improve joint situational awareness. Additionally, the usability of both the Sharemind MPC and Pleak technologies were verified as a part of work carried out and feedback provided for both.

5 CONCLUSIONS

Our work in NAPLES has lead to a wide array of modeling languages and privacy analyses for different kinds of business processes, and related executable formalisms. Our Privacy-Enhanced BPMN has been a useful modeling tool for systems with significant and complex privacy implications. Our qualitative and quantitative privacy definitions have helped to explain the nature of these implications, and our analyses have been able to precisely state, how much a system may be leaking. Regarding the definitions, a particularly important contribution has been that of *derivative sensitivity*, which clearly shows how the privacy definition is part of the privacy policy of a system, and how the latter always has to state the combinations of, and trade-offs between the leakages in various components of the data.

Almost all of our analyses have been integrated into the Pleak toolset, giving them a uniform interface for modeling the processes, specifying the privacy policies and other input parameters of the analysis, and studying the output of the analysis. The business processes can be implemented either in the PE-BPMN notation, or in plain BPMN with annotations on tasks and message flows. Pleak allows its users to inspect and analyze privacy leakages at three levels of abstraction. The first level of abstraction (the so-called "Boolean" level) allows us to see which input data sources are directly or indirectly disclosed to each party in the process. The second level (the so-called "Leaks-when" level) shows under what conditions each disclosure occurs, and which specific attributes (or functions over the input attributes) are disclosed. The third level (the "quantitative" level) allows one to measure the extent to which the disclosure reveals information about individual items (e.g. individual rows) of the input data sources. This third level also allows us to quantify to what extent the disclosure of the outputs of the process increase the probability that an attacker can guess the value of an individual item in the data source, relative to a prior probability capturing generally available knowledge about these data sources. This "attacker's guessing advantage" model also takes into account the effect of noise added to the input data in order to achieve a given differential privacy level (cf. the ϵ parameter in differential privacy).

The Pleak toolset has been validated in the DARPA Brandeis program, modeling and analysing the systems proposed and studied by the three Collaborative Research Teams. This validation provides initial evidence that the toolset can be used to analyze realistic processes. However, the ecological validity of this validation effort is limited due to the low number of processes analyzed, and by the fact that the processes have been modeled and analyzed by the same team that developed the toolset itself (even though the processes were scoped and designed by other teams in the Brandeis project). A direction for future work is to supplement this validation with usability evaluations involving business process analysts, as well as case studies conducted by independent research teams.

In its current implementation, the Pleak toolset supports the quantitative analysis of privacy-enhanced business processes in which the computations are specified in the SQL query language. In practice though, computations may be specified in various programming languages. Another avenue for future work is to extend the set of techniques in the Pleak toolset with program analysis techniques that would allow it to handle other languages.

The Pleak toolset is also limited in terms of the range of PETs it supports. In particular, the toolset does not integrate various metrics from the field of statistical disclosure control [74], including k-anonymity and l-diversity, as well as data masking techniques such as microaggregation and data swapping. Extending Pleak in order to support a wider range of PETs is another direction for future work, which we plan to pursue as a part of technology transfer efforts.

BIBLIOGRAPHY

[1] von Scheel, Henrik; von Rosing, Mark; Fonseca, Marianne; Hove, Maria; and Foldager, Ulrik, "Phase 1: Process Concept Evolution," in von Rosing, Mark; von Scheel, Henrik; and Scheer, August-Wilhelm, editors, *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM*, Morgan Kaufmann, volume 1, 2014, pp. 1–10.

[2] OMG, "Business Process Model and Notation (BPMN)," URL: http://www.omg.org/spec/BPMN/2.0/.

[3] Pullonen, Pille; Matulevičius, Raimundas; and Bogdanov, Dan, "PE-BPMN: privacy-enhanced business process model and notation," in *International Conference on Business Process Management*, Springer, pp. 40–56.

[4] Pullonen, Pille; Tom, Jake; Matulevičius, Raimundas; and Toots, Aivo, "Privacy-enhanced BPMN: enabling data privacy analysis in business processes models," *Software and Systems Modeling*, **18**(6):(2019), pp. 3235–3264, URL https://doi.org/10.1007/s10270-019-00718-z.

[5] Danezis, G.; Domingo-Ferrer, J.; Hansen, M.; Hoepman, J.-H.; Metayer, D. L.; Tirtea, R.; and Schiffner, S., "Privacy and Data Protection by Design-from Policy to Engineering," Technical report, European Union Agency for Network and Information Security, 2015.

[6] Heurix, Johannes; Zimmermann, Peter; Neubauer, Thomas; and Fenz, Stefan, "A taxonomy for privacy enhancing technologies," *Computers & Security*, **53**:(2015), pp. 1–17.

[7] Clements, Tim, "GDPR data flow mapping - an approach," , 2017, https://www.linkedin.com/pulse/gdpr-data-flow-mapping-approach-tim.

[8] Archer, David W.; Bogdanov, Dan; Pinkas, Benny; and Pullonen, Pille, "Maturity and Performance of Programmable Secure Computation," *IEEE Security and Privacy*, **14**(5):(2016), pp. 48–56.

[9] Shamir, Adi, "How to Share a Secret," *Communications of the ACM*, **22**(11):(1979), pp. 612–613, URL http://doi.acm.org/10.1145/359168.359176.

[10] Blakley, George Robert, "Safeguarding Cryptographic Keys," in *Proceedings of the 1979 AFIPS National Computer Conference*, AFIPS Press, pp. 313–317.

[11] Gilboa, Niv and Ishai, Yuval, "Distributed Point Functions and Their Applications," in *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pp. 640–658, URL http://dx.doi.org/10.1007/978-3-642-55220-5_35.

[12] Boyle, Elette; Gilboa, Niv; and Ishai, Yuval, "Function Secret Sharing," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pp. 337–367, URL http://dx.doi.org/10.1007/978-3-662-46803-6_12.

[13] Boyle, Elette; Gilboa, Niv; and Ishai, Yuval, "Breaking the Circuit Size Barrier for Secure Computation Under DDH," in *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pp. 509–539, URL http://dx.doi.org/10.1007/978-3-662-53018-4_19.

[14] Boyle, Elette; Gilboa, Niv; and Ishai, Yuval, "Function Secret Sharing: Improvements and Extensions," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pp. 1292–1303, URL http://doi.acm.org/10.1145/2976749.2978429.

[15] Shannon, Claude E, "Communication theory of secrecy systems," *Bell system technical journal*, **28**(4):(1949), pp. 656–715.

[16] Diffie, W. and Hellman, M., "New Directions in Cryptography," *IEEE Trans. Inf. Theor.*, **22**(6):(2006), pp. 644–654, URL http://dx.doi.org/10.1109/TIT.1976.1055638.

[17] Gentry, Craig, "Fully Homomorphic Encryption Using Ideal Lattices," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ACM, New York, NY, USA, STOC '09, pp. 169–178, URL http://doi.acm.org/10.1145/1536414.1536440.

[18] Paillier, Pascal, "Public-key Cryptosystems Based on Composite Degree Residuosity Classes," in *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, Springer-Verlag, Berlin, Heidelberg, EUROCRYPT'99, pp. 223–238, URL http://dl.acm.org/citation.cfm?id=1756123.1756146.

[19] Anati, Ittai; Gueron, Shay; Johnson, Simon; and Scarlata, Vincent, "Innovative technology for CPU based attestation and sealing," Technical report, Intel Corporation, 8 2013, presented at the Second Workshop on Hardware and Architectural Support for Security and Privacy, HASP 2013.

[20] "Intel Software Guard Extensions (Intel SGX)," , March 2017, https://software.intel.com/en-us/sgx.

[21] Johnson, Simon; Scarlata, Vincent; Rozas, Carlos; Brickell, Ernie; and Mckeen, Frank, "Intel software guard extensions: EPID provisioning and attestation services," *White Paper*, https://software.intel.com/en-us/blogs/2016/03/09/intel-sgx-epid-provisioning-and-attestation-services.

[22] "Intel Software Guard Extensions Remote Attestation End-to-End Example," , July 2016, https://software.intel.com/en-us/articles/intel-software-guard-extensions-remote-attestation-end-to-end-example.

[23] Belluccini, Sara; Nicola, Rocco De; Dumas, Marlon; Pullonen, Pille; Re, Barbara; ; and Tiezzi, Francesco, "Verification of privacy-enhanced collaborations," in *FormaliSE@ICSE 2020: 8th International Conference on Formal Methods in Software Engineering, Seoul, Republic of Korea, July 13, 2020*, pp. 141–152.

[24] Bergstra, Jan A and Klop, Jan Willem, "Process algebra for synchronous communication," *Information and control*, **60**(1-3):(1984), pp. 109–137.

[25] Bunte, Olav; Groote, Jan Friso; Keiren, Jeroen JA; Laveaux, Maurice; Neele, Thomas; de Vink, Erik P; Wesselink, Wieger; Wijs, Anton; and Willemse, Tim AC, "The mCRL2 Toolset for Analysing Concurrent Systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Springer, pp. 21–39.

[26] Groote, Jan Friso and Mousavi, Mohammad Reza, *Modeling and analysis of communicating systems*, MIT press, 2014.

[27] Tšahhirov, Ilja and Laud, Peeter, "Application of Dependency Graphs to Security Protocol Analysis," in Barthe, Gilles and Fournet, Cédric, editors, *Trustworthy Global Computing, Third Symposium, TGC 2007, Sophia-Antipolis, France, November 5-6, 2007, Revised Selected Papers*, Springer, volume 4912 of *Lecture Notes in Computer Science*, pp. 294–311, URL https://doi.org/10.1007/978-3-540-78663-4_20.

[28] Laud, Peeter and Tšahhirov, Ilja, "A User Interface for a Game-Based Protocol Verification Tool," in Degano, Pierpaolo and Guttman, Joshua D., editors, *Formal Aspects in Security and Trust, 6th International Workshop, FAST 2009, Eindhoven, The Netherlands, November 5-6, 2009, Revised Selected Papers*, Springer, volume 5983 of *Lecture Notes in Computer Science*, pp. 263–278, URL https://doi.org/10.1007/978-3-642-12459-4_19.

[29] Dijkman, Remco M.; Dumas, Marlon; and Ouyang, Chun, "Semantics and analysis of business process models in BPMN," *Information & Software Technology*, **50**(12):(2008), pp. 1281–1294, URL https://doi.org/10.1016/j.infsof.2008.02.006.

[30] Dumas, Marlon; García-Bañuelos, Luciano; and Laud, Peeter, "Disclosure Analysis of SQL Workflows," in Cybenko, George; Pym, David J.; and Fila, Barbara, editors, *5th International Workshop on Graphical Models for Security, held in conjunction with the Federated Logic Conference (FLoC) 2018, GraMSec@FLoC 2018, Oxford, UK, July 8, 2018, Revised Selected Papers*, Springer, volume 11086 of *Lecture Notes in Computer Science*, pp. 51–70, URL https://doi.org/10.1007/978-3-030-15465-3_4.

[31] Esparza, Javier; Römer, Stefan; and Vogler, Walter, "An Improvement of McMillan's Unfolding Algorithm," *Formal Methods in System Design*, **20**(3):(2002), pp. 285–310, URL https://doi.org/10.1023/A:1014746130920.

[32] Armas-Cervantes, Abel; Baldan, Paolo; Dumas, Marlon; and García-Bañuelos, Luciano, "Diagnosing behavioral differences between business process models: An approach based on event structures," *Inf. Syst.*, **56**:(2016), pp. 304–325, URL https://doi.org/10.1016/j.is.2015.09.009.

[33] Milner, Robin, *Communicating and mobile systems - the Pi-calculus*, Cambridge University Press, 1999.

[34] Fournet, Cédric and Gonthier, Georges, "The Reflexive CHAM and the Join-Calculus," in Boehm, Hans-Juergen and Jr., Guy L. Steele, editors, *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996*, ACM Press, pp. 372–385, URL http://doi.acm.org/10.1145/237721.237805.

[35] Laud, Peeter and Randmets, Jaak, "A Domain-Specific Language for Low-Level Secure Multiparty Computation Protocols," in Ray, Indrajit; Li, Ninghui; and Kruegel, Christopher, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, ACM, pp. 1492–1503, URL http://doi.acm.org/10.1145/2810103.2813664.

[36] Dwork, Cynthia, "Differential Privacy," in Bugliesi, Michele; Preneel, Bart; Sassone, Vladimiro; and Wegener, Ingo, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, Springer, volume 4052 of *Lecture Notes in Computer Science*, pp. 1–12, URL http://dx.doi.org/10.1007/11787006_1.

[37] Chatzikokolakis, Konstantinos; Andrés, Miguel E.; Bordenabe, Nicolás Emilio; and Palamidessi, Catuscia, "Broadening the Scope of Differential Privacy Using Metrics," in Cristofaro, Emiliano De and Wright, Matthew, editors, *Privacy Enhancing Technologies - 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings*, Springer, volume 7981 of *Lecture Notes in Computer Science*, pp. 82–102, URL http://dx.doi.org/10.1007/978-3-642-39077-7_5.

[38] Ebadi, Hamid; Sands, David; and Schneider, Gerardo, "Differential Privacy: Now it's Getting Personal," in Rajamani, Sriram K. and Walker, David, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai,*

India, January 15-17, 2015, ACM, pp. 69–81, URL http://doi.acm.org/10.1145/2676726.2677005.

[39] Cousot, Patrick and Cousot, Radhia, "Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints," in Graham, Robert M.; Harrison, Michael A.; and Sethi, Ravi, editors, *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, ACM, pp. 238–252, URL http://doi.acm.org/10.1145/512950.512973.

[40] Barthe, Gilles; D'Argenio, Pedro R.; and Rezk, Tamara, "Secure Information Flow by Self-Composition," in *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*, IEEE Computer Society, pp. 100–114, URL http://doi.ieeecomputersociety.org/10.1109/CSFW.2004.17.

[41] Bogdanov, Dan; Laud, Peeter; and Randmets, Jaak, "Domain-Polymorphic Programming of Privacy-Preserving Applications," in Russo, Alejandro and Tripp, Omer, editors, *Proceedings of the Ninth Workshop on Programming Languages and Analysis for Security, PLAS@ECOOP 2014, Uppsala, Sweden, July 29, 2014*, ACM, pp. 53–65, URL http://doi.acm.org/10.1145/2637113.2637119.

[42] Bogdanov, Dan; Kamm, Liina; Kubo, Baldur; Rebane, Reimo; Sokk, Ville; and Talviste, Rivo, "Students and Taxes: a Privacy-Preserving Study Using Secure Computation," *PoPETs*, **2016**(3):(2016), pp. 117–135, URL http://www.degruyter.com/view/j/popets.2016.2016.issue-3/popets-2016-0019/popets-2016-0019.xml.

[43] Bogdanov, Dan; Niitsoo, Margus; Toft, Tomas; and Willemson, Jan, "High-performance secure multi-party computation for data mining applications," *Int. J. Inf. Sec.*, **11**(6):(2012), pp. 403–418, URL http://dx.doi.org/10.1007/s10207-012-0177-2.

[44] Karr, Michael, "Affine Relationships Among Variables of a Program," *Acta Inf.*, **6**:(1976), pp. 133–151, URL http://dx.doi.org/10.1007/BF00268497.

[45] Cousot, Patrick and Halbwachs, Nicolas, "Automatic Discovery of Linear Restraints Among Variables of a Program," in Aho, Alfred V.; Zilles, Stephen N.; and Szymanski, Thomas G., editors, *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages, Tucson, Arizona, USA, January 1978*, ACM Press, pp. 84–96, URL http://doi.acm.org/10.1145/512760.512770.

[46] Granger, Philippe, "Static Analysis of Linear Congruence Equalities among Variables of a Program," in Abramsky, Samson and Maibaum, T. S. E., editors, *TAPSOFT'91: Proceedings of the International Joint Conference on Theory and Practice of Software Development, Brighton, UK, April 8-12, 1991, Volume 1: Colloquium on Trees in Algebra and Programming (CAAP'91)*, Springer, volume 493 of *Lecture Notes in Computer Science*, pp. 169–192, URL http://dx.doi.org/10.1007/3-540-53982-4_10.

[47] Laud, Peeter, *Computationally Secure Information Flow*, Ph.D. thesis, University of Saarland, 2002.

[48] Gover, Thomas M. and Thomas, Joy A., *Elements of Information Theory*, John Wiley & Sons, 2nd edition, 2006.

[49] Cuff, Paul and Yu, Lanqing, "Differential Privacy as a Mutual Information Constraint," in Weippl, Edgar R.; Katzenbeisser, Stefan; Kruegel, Christopher; Myers, Andrew C.; and Halevi, Shai, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, ACM, pp. 43–54, URL https://doi.org/10.1145/2976749.2978308.

[50] Alvim, Mário S.; Andrés, Miguel E.; Chatzikokolakis, Konstantinos; Degano, Pierpaolo; and Palamidessi, Catuscia, "On the information leakage of differentially-private mechanisms," *Journal of Computer Security*, **23**(4):(2015), pp. 427–469, URL http://dx.doi.org/10.3233/JCS-150528.

[51] Kopperman, Ralph, "All Topologies Come From Generalized Metrics," *The American Mathematical Monthly*, **95**(2):(1988), pp. 89–97, URL http://www.jstor.org/stable/2323060.

[52] Nissim, Kobbi; Raskhodnikova, Sofya; and Smith, Adam D., "Smooth sensitivity and sampling in private data analysis," in Johnson, David S. and Feige, Uriel, editors, *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, ACM, pp. 75–84, URL http://doi.acm.org/10.1145/1250790.1250803.

[53] Guagliardo, Paolo and Libkin, Leonid, "A Formal Semantics of SQL Queries, Its Validation, and Applications," *Proceedings of the VLDB Endowment (PVLDB)*.

[54] McSherry, Frank, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," in Çetintemel, Ugur; Zdonik, Stanley B.; Kossmann, Donald; and Tatbul, Nesime, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, ACM, pp. 19–30, URL http://doi.acm.org/10.1145/1559845.1559850.

[55] Baggett, L.W., *Functional Analysis: A Primer*, Chapman & Hall Pure and Applied Mathematics, Taylor & Francis, 1991, URL https://books.google.ee/books?id=RKFiQgAACAAJ.

[56] He, Xi; Machanavajjhala, Ashwin; and Ding, Bolin, "Blowfish privacy: tuning privacy-utility trade-offs using policies," in Dyreson, Curtis E.; Li, Feifei; and Özsu, M. Tamer, editors, *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, ACM, pp. 1447–1458, URL http://doi.acm.org/10.1145/2588555.2588581.

[57] Geng, Quan; Ding, Wei; Guo, Ruiqi; and Kumar, Sanjiv, "Truncated Laplacian Mechanism for Approximate Differential Privacy," *CoRR*, **abs/1810.00877**, URL http://arxiv.org/abs/1810.00877.

[58] Lee, Jaewoo and Clifton, Chris, "How Much Is Enough? Choosing ϵ for Differential Privacy," in Lai, Xuejia; Zhou, Jianying; and Li, Hui, editors, *Information Security, 14th International Conference, ISC 2011, Xi'an, China, October 26-29, 2011. Proceedings*, Springer, volume 7001 of *Lecture Notes in Computer Science*, pp. 325–340, URL https://doi.org/10.1007/978-3-642-24861-0_22.

[59] Kifer, Daniel and Machanavajjhala, Ashwin, "Pufferfish: A Framework for Mathematical Privacy Definitions," *ACM Trans. Database Syst.*, **39**(1):(2014), pp. 3:1–3:36, URL http://doi.acm.org/10.1145/2514689.

[60] Oh, Sewoong and Viswanath, Pramod, "The Composition Theorem for Differential Privacy," *CoRR*, **abs/1311.0776**, URL http://arxiv.org/abs/1311.0776.

[61] Sommer, David M.; Meiser, Sebastian; and Mohammadi, Esfandiar, "Privacy Loss Classes: The Central Limit Theorem in Differential Privacy," *PoPETs*, **2019**(2):(2019), pp. 245–269, URL https://doi.org/10.2478/popets-2019-0029.

[62] Balle, Borja and Wang, Yu-Xiang, "Improving the Gaussian Mechanism for Differential Privacy: Analytical Calibration and Optimal Denoising," in Dy, Jennifer G. and Krause, Andreas, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, PMLR, volume 80 of *Proceedings of Machine Learning Research*, pp. 403–412, URL http://proceedings.mlr.press/v80/balle18a.html.

[63] Stolfi, Jorge and de Figueiredo, Luiz Henrique, "An introduction to affine arithmetic," *Trends in Applied and Computational Mathematics*, **4**(3):(2003), pp. 297–312.

[64] Stolfi, Jorge and Figueiredo, Luiz Henrique De, "Self-validated numerical methods and applications," , 1997, set of course notes for the 21st Brazilian Mathematics Colloquium, IMPA, Rio de Janeiro.

[65] Gay, Olivier; Coeurjolly, David; and Hurst, Nathan, "Libaffa-C++ affine arithmetic library for GNU/Linux," , 2006.

[66] Fan, Liyue and Xiong, Li, "Adaptively Sharing Time-Series with Differential Privacy," *CoRR*, **abs/1202.3461**, URL http://arxiv.org/abs/1202.3461.

[67] Dwork, Cynthia; Naor, Moni; Pitassi, Toniann; and Rothblum, Guy N., "Differential privacy under continual observation," in Schulman, Leonard J., editor, *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, ACM, pp. 715–724, URL https://doi.org/10.1145/1806689.1806787.

[68] Toots, Aivo; Tuuling, Reedik; Yerokhin, Maksym; Dumas, Marlon; García-Bañuelos, Luciano; Laud, Peeter; Matulevičius, Raimundas; Pankova, Alisa; Pettai, Martin; Pullonen, Pille; and Tom, Jake, "Business Process Privacy Analysis in Pleak," in Hähnle, Reiner and van der Aalst, Wil M. P., editors, *Fundamental Approaches to Software Engineering - 22nd International Conference, FASE 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, Springer, volume 11424 of *Lecture Notes in Computer Science*, pp. 306–312, URL https://doi.org/10.1007/978-3-030-16722-6_18.

[69] Lepinski, Matthew; Levin, David; McCarthy, Daniel; Watro, Ronald; Lack, Michael; Hallenbeck, Daniel; and Slater, David, "Privacy-enhanced android for smart cities applications," in *Smart City 360*, Springer, 2016, pp. 66–77.

[70] Jeannet, Bertrand and Miné, Antoine, "Apron: A Library of Numerical Abstract Domains for Static Analysis," in Bouajjani, Ahmed and Maler, Oded, editors, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, Springer, volume 5643 of *Lecture Notes in Computer Science*, pp. 661–667, URL https://doi.org/10.1007/978-3-642-02658-4_52.

[71] "TPC BENCHMARK™ H, revision 2.17.3," Transaction Processing Performance Council, 2017, http://www.tpc.org/TPC_Documents_Current_Versions/pdf/tpc-h_v2.17.3.pdf.

[72] Chen, Yan; Machanavajjhala, Ashwin; Hay, Michael; and Miklau, Gerome, "PeGaSus: Data-Adaptive Differentially Private Stream Processing," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ACM, New York, NY, USA, CCS '17, pp. 1375–1388, URL http://doi.acm.org/10.1145/3133956.3134102.

[73] Virtanen, Pauli; Gommers, Ralf; Oliphant, Travis E.; Haberland, Matt; Reddy, Tyler; Cournapeau, David; Burovski, Evgeni; Peterson, Pearu; Weckesser, Warren; Bright, Jonathan; van der Walt, Stéfan J.; Brett, Matthew; Wilson, Joshua; Jarrod Millman, K.; Mayorov, Nikolay; Nelson, Andrew R. J.; Jones, Eric; Kern, Robert; Larson, Eric; Carey, CJ; Polat, İlhan; Feng, Yu; Moore, Eric W.; Vand erPlas, Jake; Laxalde, Denis; Perktold, Josef; Cimrman, Robert; Henriksen, Ian; Quintero, E. A.; Harris, Charles R; Archibald, Anne M.; Ribeiro, Antônio H.; Pedregosa, Fabian; van Mulbregt, Paul; and Contributors, SciPy 1. 0, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*.

[74] Hundepool, Anco; Domingo-Ferrer, Josep; Franconi, Luisa; Giessing, Sarah; Nordholt, Eric Schulte; Spicer, Keith; and de Wolf, Peter-Paul, *Statistical Disclosure Control*, Wiley, 2012.

A LIST OF PUBLICATIONS

The following lists the publications (research papers, technical reports and blog posts) that have appeared so far as part of the NAPLES project.

- Marlon Dumas, Luciano García-Bañuelos, Peeter Laud: *Differential Privacy Analysis of Data Processing Workflows*. GraMSec@CSF 2016: 62-79
- Martin Pettai, Peeter Laud: *Combining Differential Privacy and Mutual Information for Analyzing Leakages in Workflows*. POST 2017: 298-319
- Pille Pullonen, Raimundas Matulevičius, Dan Bogdanov: *PE-BPMN: Privacy-Enhanced Business Process Model and Notation*. BPM 2017: 40-56 https://link.springer.com/chapter/10.1007/978-3-319-65000-5_3
- Marlon Dumas, Luciano García-Bañuelos, Peeter Laud: *Disclosure Analysis of SQL Workflows*. GraMSec@CSF 2018: 51-70
- Pille Pullonen, Jake Tom, Raimundas Matulevičius, Aivo Toots: *Privacy-enhanced BPMN: enabling data privacy analysis in business processes models*. Software & Systems Modeling, 2019 https://link.springer.com/article/10.1007/s10270-019-00718-z
- Aivo Toots, Reedik Tuuling, Maksym Yerokhin, Marlon Dumas, Luciano Garcia-Banuelos, Peeter Laud, Raimundas Matulevicius, Alisa Pankova, Martin Pettai, Pille Pullonen, Jake Tom: *Business Process Privacy Analysis in Pleak*. FASE, 2019. Full version available in https://arxiv.org/abs/1902.05052
- Aivo Toots, Reedik Tuuling, Maksym Yerokhin, Marlon Dumas, Luciano García-Bañuelos, Peeter Laud, Raimundas Matulevicius, Alisa Pankova, Martin Pettai, Pille Pullonen, Jake Tom: *Business Process Privacy Analysis in Pleak - (Extended Abstract)*. Inform. Spektrum 42(5): 354-355 (2019)
- Pille Pullonen: *How privacy enhancing technologies affect business processes*. Sharemind blog 2019 https://sharemind.cyber.ee/studying-processes-with-privacy-technologies/
- Peeter Laud, Alisa Pankova, Martin Pettai: *Achieving Differential Privacy using Methods from Calculus*, https://arxiv.org/abs/1811.06343
- Peeter Laud, Alisa Pankova: *Interpreting Epsilon of Differential Privacy in Terms of Advantage in Guessing or Approximating Sensitive Attributes*, https://arxiv.org/abs/1911.12777
- Sara Belluccini, Rocco De Nicola, Marlon Dumas, Pille Pullonen, Barbara Re, and Francesco Tiezzi: *Verification of privacy-enhanced collaborations*. FormaliSE 2020
- Pille Pullonen: *Analysing Privacy of Business Processes in Pleak*. Cybernetica's blog 2020 https://cyber.ee/blog/2020/04-20/

B LIST OF SYMBOLS AND ACRONYMS

$Ber(p)$	Bernoulli distribution with parameter p
DS_f	derivative sensitivity of a function f
GS_f	global sensitivity of a function f
$GenCauchy(\gamma)$	generalized Cauchy distribution with parameter γ
LS_f	local sensitivity of a function f
$Lap(\lambda)$	Laplace distribution with scaling λ
$N(\mu, \sigma^2)$	Normal distribution with mean μ and standard deviation σ
O	Big O; Big Oh; Big Omicron
$[m]$	the set of integers $\{1, \ldots, m\}$
$[x]_\rho$	equivalence class w.r.t. relation ρ
Δ	difference
Ω	Big Omega
Θ	Big Theta
$\boxtimes$	the set of probability distributions over pairs, whose marginal distributions are the arguments of the operation
δ_{xy}	Kronecker symbol, equal to 1 iff $x = y$, and 0 otherwise
$\downarrow$	the downwards closure of a subset of a partially ordered set
η	monadic unit (in category theory)
$\exists$	existential quantifier
$\forall$	universal quantifier
$\frac{\partial f}{\partial x}$	partial derivative of a function f w.r.t. variable x
$\int_X$	(Lebesque) integration over the set X
$\mathbb{B}$	set $\{0, 1\}$
$\mathbb{N}$	set of natural numbers
$\mathbb{R}$	set of real numbers
$\mathbb{R}_+$	set of positive real numbers
$\mathbb{Z}$	set of integers
$\mathcal{D}$	the set of all probability distributions over the given set
$\mathcal{F}$	the set of all upwards closed subsets of the argument
$\mathcal{I}$	the set of all downwards closed subsets of the argument
$\mathcal{M}_f$	a privacy enhancing mechanism applied to f
Carr	carrier set of the given generalized metric space
Pr	probability
c	the map between sets of distances for a mapping between generalized metric spaces
f	the map between carrier sets for a mapping between generalized metric spaces
d_{dp}	the *differential privacy distance* between probability distributions
supp	the support of a probability distribution
μ	monadic multiplication (in category theory)

∇f	gradient of a function f
$\neg$	negation (not)
$\preceq$	if N and M are norms, $N \preceq M$ denotes that $\|\vec{x}\|_N \leq \|\vec{x}\|_M$ for all $\vec{x}$
$\prod$	product
$\sum$	sum
$\uparrow$	the upwards closure of a subset of a partially ordered set
$\vec{x}$	vector
$\vee$	disjunction (or)
$\wedge$	conjunction (and)
d_X	distance in the set X
df_x	Fréchet derivative of f at the point x
f_X	probability density function of a random variable X
$\|\vec{x}\|_p$	ℓ_p-norm of a vector $\vec{x}$

AddSS	additive secret sharing
AES	Advanced Encryption Standard
AFRL	Air Force Research Laboratory
BDD	binary decision diagram
BP	Business Process
BPMN	Business Process Model and Notation
CDF	cumulative density function
CPU	Central Processing Unit
CRT	collaborative research team
D	direct dependency (in disclosure analysis)
DAG	directed acyclic graph
DARPA	Defense Advanced Research Projects Agency
DG	dependency graph
DP	differential privacy
DP-Workflow	data processing workflow
FHE	fully homomorphic encryption
FunSS	function secret sharing
GA	guessing advantage
GC	garbled circuit
GM	the category of generalized metric spaces
GPS	Global Positioning System
H	entropy (in quantitative analysis)
H	hidden (in disclosure analysis)
I	mutual information (in quantitative analysis)
I	indirect dependency (in disclosure analysis)
ID	identity

IoT	Internet of Things
IPSec	Internet Protocol Security
IT	Information Technology
KDE	kernel density estimation
MAC	Medium Access Control
MPC	Secure multiparty computation
NAPLES	Novel Tools for Analyzing Privacy Leakages
O	owner (in disclosure analysis)
OT	oblivious transfer
PA	project agreement
PAL	privacy abstraction layer
PDF	probability density function
PE	privacy enhanced
PE-BPMN	Privacy Enhanced Business Process Model and Notation
PET	privacy enhancing technology
PINQ	Privacy INtegrated Queries – a platform for privacy-preserving data analysis
PK	public key
PRESNA	Privacy Enhanced Social Network Analysis
PSDG	partial summary dependency graph
PULSAR	Private Updateable Lightweight Scalable Active Repository
RAM	Random Access Memory
REST	representational state transfer
SDG	summary dependency graph
SF	scale factor
SGX	Software Guard Extension
SIMD	single instruction multiple data
SIRD	disease states *Suspectible*, *Infected*, *Recovered*, *Deceased* in the Pandemic scenario
SK	secret key
SMT	satisfiablity *modulo* theories
SoD	the set of distances of the given generalized metric space
SQL	Structured Query Language
SS	secret sharing
TA	technical area
TIPPERS	Testbed for IoT-based Privacy-Preserving PERvasive Spaces
TTL	time-to-live
UCI	University of California, Irvine
V	visible (in disclosure analysis)
VPN	Virtual Private Network