Differential Privacy for Probabilistic Systems

Michael Carl Tschantz Anupam Datta Dilsun Kaynar

May 14, 2009
CMU-CyLab-09-008

CyLab
Carnegie Mellon University
Pittsburgh, PA 15213

Differential Privacy for Probabilistic Systems*

Michael Carl Tschantz
mtschant@cs.cmu.edu

Anupam Datta
danupam@cmu.edu

Dilsun Kaynar
dilsunk@cmu.edu

Abstract

Differential privacy is a promising approach to privacy-preserving data analysis. There is now a well-developed theory of differentially private functions. Despite recent work on implementing database systems that aim to provide differential privacy and distributed systems that use differential privacy as a basis for higher level security properties, there is no formal theory of differential privacy for *systems*. In this paper, we formulate precise definitions of differential privacy within a formal model of probabilistic systems, relate these definitions to the original definitions, and develop a proof technique based on an *unwinding relation* for establishing that a given system achieves this privacy definition. We illustrate the proof technique on a representative example motivated by an implemented system.

1 Introduction

Differential privacy is a promising approach to privacy-preserving data analysis (see [9] for a survey). This work is motivated by statistical data sets that contain personal information about a large number of individuals, e.g., census or health data. In such a scenario, a trusted party collects personal information from a representative sample with the goal of releasing statistics about the underlying population while simultaneously protecting the privacy of individuals. In an interactive setting, an untrusted data analyst poses queries that are evaluated over the data set and appropriately modified by the trusted party before the response is sent back to the analyst. Differential privacy formalizes this operation in terms of a probabilistic function that takes the data set as input. Informally, the differential privacy guarantee says that the output of the function does not change much irrespective of whether any particular individual is part of the data set or not. The amount of change is measured in terms of a *privacy error bound*—a non-negative real number ϵ, where a smaller ϵ indicates a higher level of privacy. In practice, one function is associated with each type of query on the data set. The definition and privacy guarantees extend in a natural way to a setting with multiple functions. The insight here is that since only a limited amount of additional privacy risk is incurred by joining a data set, individuals may decide to join the data set if there are social benefits from doing so (e.g., aid cancer research). A consequence and strength of the definition is that the privacy guarantee holds irrespective of the auxiliary information and computational power available to an adversary. There is now a rich body of work on *differentially private functions*, including provably secure constructions and sensitivity analysis of such functions, trade-offs between privacy and utility (answering useful queries), and related questions [7, 18, 20, 3, 8, 9].

In a different direction, there is also some early work in implementing database systems that aim to provide differential privacy [17] as well as distributed systems that use differential privacy as a basis for releasing privacy-preserving information [1]. The definition of differential privacy for functions has to be refined to adequately capture properties of such systems. In addition, proving properties of such system designs and implementations require very different techniques. In order to bridge this gap, we initiate a systematic study of *differential privacy for systems* in this paper. We formulate precise definitions of *trace*

*This work was partially supported by the U.S. Army Research Office contract on Perpetually Available and Secure Information Systems (DAAD19-02-1-0389) to CMU's CyLab, the NSF Science and Technology Center TRUST, and the NSF CyberTrust grant "Privacy, Compliance and Information Risk in Complex Organizational Processes".

differential privacy within a formal model of probabilistic systems (Section 2), relate these definitions to the original definitions (Section 3), develop a proof technique for establishing that a given system achieves trace differential privacy (Section 4), and illustrate the proof technique on a representative example motivated by the PINQ system [17] (Section 4.3). We elaborate on these contributions below.

Trace Differential Privacy. We introduce the notion of *trace differential privacy* and formalize it within a probabilistic model of systems. (The word "trace" is included in the name of the property because it is formulated in terms of probability distributions over traces (or executions) of systems.) At a high-level, the definition requires that the probability distributions over traces are approximately the same when the implementation of a differentially private function (modeled as an automata) is executed with inputs that differ on one data point. We consider two variants of trace differential privacy—*strong* and *weak*. The difference between the two is that the strong version requires the privacy error bound to be a fixed constant (analogous to [17]), whereas the weak version allows the bound to vary depending on the number and types of queries posed (analogous to [7]).

Unlike the original definition of differential privacy, these definitions take into account two specific issues that arise in real systems—*asynchrony* and *mutable data sets*. Since an asynchronous system may not answer queries in the order in which they are posed (e.g., in order to improve performance), it has additional channels for leaking private information. Also, data sets that change over time arise quite often in practice (e.g., the data set of patient health records in a hospital) and raise challenges for obtaining tight privacy error bounds.

We prove some basic theorems about strong and weak trace differential privacy in Section 2. Although the definitions of trace differential privacy refer to all possible subsets of the set of traces, Theorems 1-2 prove that they are equivalent to definitions that refer to each trace separately. This result simplifies reasoning about trace differential privacy and is useful for proving subsequent results in this paper. Theorems 3-4 are composition theorems that are analogous to previous results about differential privacy for functions: they prove that the privacy error bound for a system whose inputs differ on at most n data points is $n * \epsilon$, where ϵ is the error bound for the system if its inputs differ on one data point.

Relating the Definitions. We prove several theorems (Theorems 5-8) characterizing the relationship between the original definition of differential privacy for functions and our definitions of trace differential privacy.

The first set of results indicate that trace differential privacy is a conservative extension of differential privacy for functions. Specifically, Theorem 5 roughly states that if an automaton K has weak trace differential privacy, then K treated as a function on data sets will have differential privacy. However, to treat K as a function and to avoid problems in the original definition of differential privacy involving non-measurable sets, we require that K is *terminating*. Furthermore, in order to obtain the desired privacy error bound, we require the technical condition that termination is *noticeable*, i.e. the end of a trace is explicitly marked using a distinguished action. The termination or noticeable termination condition is not required to prove a similar theorem (Theorem 6) about a variant of differential privacy that we call *prefix differential privacy*.

We then investigate the problem of constructing a system that satisfies trace differential privacy from functions that individually satisfy differential privacy. Such results are useful because they allow us to use existing constructions for differentially private functions to achieve trace differential privacy. We prove that a precisely defined class of automata that are *synchronous database implementations* of a set of functions satisfy weak trace differential privacy (Theorem 7). In effect, such an automaton answers queries right after they are posed and independently of any other queries that it receives. In contrast, we show via examples that asynchrony in real systems has subtle implications for trace differential privacy. The first example shows that the order in which the queries are answered (as determined by a query scheduler) may leak information about a data point and violate our definitions even if we use differentially private functions. The second example indicates that even if the query scheduler and the differentially private function do not depend on data points, private information can be leaked based on the dependence between the scheduler and the function; the attack, however, violates the definition of trace differential privacy, providing additional evidence of its robustness.

Finally, we address the problem of proving trace differential privacy guarantees for data sets that change over time. One way in which the original definition of differential privacy can handle a mutable data set is to include every data point and every query ever asked in calculating a conservative privacy error bound. In contrast, we develop a higher fidelity accounting scheme, using a sliding window over data points and queries in the input sequence. We impose the constraint that at any point the window contains at most t queries. This captures the intuition that a data point can be used to answer only up to t queries and after that it is removed from the sliding window of data points used to answer queries. It also provides a mechanism for dropping old points from the data set. We prove that in this scenario the associated automata provides $(t * \epsilon)$-*strong trace differential privacy*, where ϵ is the error bound for each differentially private function used to answer queries (Theorem 8). The parameter t can be adjusted to trade-off the privacy error bound against the subset of the full data set that is used in computing responses to queries.

Proof Technique and Example. We define an operational model of probabilistic systems using a form of probabilistic input-output automata [14]. These automata accept input sequences consisting of data points and queries and execute probabilistically to produce traces. Since it is difficult to reason directly about probability distributions over traces, we define an *unwinding relation* over such automata and prove that it implies strong trace differential privacy (Theorem 9). This task is complicated for two reasons: we must deal with probabilities and we must keep track of the privacy error bound ϵ. While previous work has dealt with approximate probabilistic simulation relations in the context of cryptographic protocols [22], the process of tracking a privacy error bound differs (see Section 5 for additional details). We illustrate this proof technique by demonstrating an unwinding relation for an automaton motivated by the PINQ system [17].

2 Differential Privacy and Systems

2.1 Differential Privacy for Functions

Differential privacy protects the privacy of data points. A data point represents all the information about a single person (or other entity that must be protected) in a data set, where a data set is a multiset of data points. Differential privacy requires that the protected data set only be accessed by way of a *sanitization function*. A sanitization function κ is a function from a data set to some information about the provided data set. Intuitively, differential privacy requires that this information does not change much if the provided data set is changed by one data point. The amount of change is measured in terms of a non-negative real number ϵ. The larger ϵ is, the greater the possible change and the less privacy provided. We state below the original formal definition of differential privacy [7].

Definition 1 (Differential Privacy)**.** *A randomized function κ gives ϵ-differential privacy iff for all data sets B_1 and B_2 differing on at most one element, and for all $S \subseteq \mathsf{range}(\kappa)$,*

$$\Pr[\kappa(B_1) \in S] \leq \exp(\epsilon) * \Pr[\kappa(B_2) \in S]$$

Formally, multisets B_1 and B_2 differ on at most one element iff either $B_1 = B_2$ or there exists d such that $B_1 \cup \{d\} = B_2$ or $B_2 \cup \{d\} = B_1$.

Note that the above definition is well-defined only if $\mathsf{range}(\kappa)$ is countable. Otherwise, there may exist $S \subseteq \mathsf{range}(\kappa)$ that is non-measurable (constructed with the axiom of choice), for which $\Pr[\kappa(B_1) \in S]$ is not well-defined.

2.2 From Functions to Systems

When we think about how sanitized databases are implemented and want to define a differential privacy notion for such systems, it is natural think about the range of κ in Definition 1 as a set of systems that answer queries about their data sets, provided that systems are appropriately constrained to form a countable set. These systems could be formalized by using, for example, automata or processes in a process calculus.

Definition 1 would then require a notion of "equality" for systems. As is typically done in concurrency theory, we base this notion on observable behaviors of systems. This subsection gives some intermediate definitions motivated by this observation. These definitions are useful to follow the line of thought that led us to our definition of differential privacy for systems, which we present in the next subsection.

If we assume that $\mathsf{range}(\kappa)$ is a system instead of a function, we can interpret ϵ in Definition 1 as determining the maximum allowable degradation in privacy. In this interpretation, κ embodies the implementation of several sanitization functions that can be used by the system as long as ϵ allows.

Probabilistic model. To arrive at a formal definition of differential privacy for systems based on the similarity of their behaviors, we first need to have a formal model for systems. Our model is based on random variables over action sequences (in Section 4.1 we present a stateful automaton model that is consistent with the model of this section).

We model the *inputs* that the system accepts as a set I and the *outputs* that it produces as a set O such that $I \cap O = \emptyset$. Let A be $I \cup O$, the set of *actions*. Since a computation consumes inputs and produces outputs by interleaving them, it might be tempting to treat a system as a function from I^* to A^*. However, a computation might not terminate. Thus, a system actually acts as a function from I^* to $A^* \cup A^\omega$. We model the observable behavior by the set of *traces* that a system exhibits, where a trace is defined to be a sequence of actions from $A^* \cup A^\omega$. Furthermore, since our system might be probabilistic, we model a system as function K that takes a sequence inputs $\vec{i}$ in I^* and returns a random variable $K(\vec{i})$ over behaviors in $A^* \cup A^\omega$. That is, $K(\vec{i})$ represents a random variable over behaviors from $A^* \cup A^\omega$ given that the available inputs are $\vec{i}$ in that order.

Since A^ω is uncountable for even finite action sets A, defining $\Pr[K(\vec{i}) \in S]$ for $S \subseteq A^* \cup A^\omega$ poses problems as mentioned above. However, this is unnecessary: an observer can only observe finite prefixes of any non-terminating computation. After observing such a prefix, the observer must consider all behaviors with the observed prefix to be possible. Thus, one may view observations of computations to be sets of traces all of which have the same finite prefix. These sets are in one-to-one correspondence with A^*. Thus, we model the result of a computation as an element $\vec{a}$ of A^* bearing in mind that the result $\vec{a}$ does not mean that the computation was $\vec{a}$, but rather that the computation has $\vec{a}$ as a prefix. Thus, rather than concerning ourselves with the probability of the system producing some behavior, we concern ourselves with the probability of a system producing a behavior with some prefix. That is, rather than $\Pr[K(\vec{i}) \in S]$ for $S \subseteq A^* \cup A^\omega$, we need $\Pr[K(\vec{i}) \sqsupseteq S]$ for $S \subseteq A^*$ where $\sqsupseteq$ is the super-sequence-equal operator raised to work over sets in the following manner: $\vec{a} \sqsupseteq S$ iff there exists $\vec{a}' \in S$ such that $\vec{a} \sqsupseteq \vec{a}'$ where $\vec{a} \in A^* \cup A^\omega$ and $S \subseteq A^*$. Since A is countable, so is A^*, allowing $\Pr[K(\vec{i}) \sqsupseteq S]$ to be well defined for all $S \subseteq A^*$ using discrete probabilities.[1] In Section 4.1, we explain how to view probabilistic automata as such a randomized function and define $\Pr[K(\vec{i}) \sqsupseteq S]$ for them.

In the following definition, κ represents a function that outputs a system. Then, $\kappa(B_1)$ represents the system that κ selects to answer queries about the data set B_1. The queries are represented by inputs from I and the answers by outputs from O. $\kappa(B_1)(\vec{i})$ is a random variable over traces from A^*.

Definition 2. *A randomized function κ gives ϵ-strong differential privacy over systems with inputs I and output O if for all data sets B_1 and B_2 differing on at most one element, and for all $S \subseteq A^*$ and $\vec{i} \in I^*$,*

$$\Pr[\kappa(B_1)(\vec{i}) \sqsupseteq S] \leq \exp(\epsilon) * \Pr[\kappa(B_2)(\vec{i}) \sqsupseteq S]$$

This is a very strong definition since it requires that the amount of information provided by the automaton $\kappa(B_1)$ does not increase beyond what is determined by the constant ϵ regardless of the number of interactions. While it is possible to construct such an automaton, many realistic systems might provide some privacy despite not having this property by slowly leaking information. Thus, we also present a version that depends on the number of queries asked.

[1] This observation is akin to the cone construction used in probabilistic automata work [14]. There, the cone of an execution α is defined as the set of all execution with the prefix α, and cones are assigned measures.

Since different queries might release different amounts of information, we must ensure that our definition takes this into account. Let $c : I \to \mathbb{R}^{\geq 0}$ be a mapping from a query (represented as an input) to the amount ϵ of differential privacy it provides. We raise c to work over sequences of inputs as follows: $c([]) = 0$ and $c(i{:}\vec{i}) = c(i) + c(\vec{i})$. The following is a weaker version of Definition 2 in which the privacy error bound depends on the number of queries asked.

Definition 3. *A randomized function κ gives c-weak differential privacy over systems with inputs I and outputs O if for all data sets B_1 and B_2 differing on at most one element, and for all $S \subseteq A^*$ and $\vec{i} \in I^*$,*

$$\Pr[\kappa(B_1)(\vec{i}) \sqsupseteq S] \leq \exp(c(\vec{i})) * \Pr[\kappa(B_2)(\vec{i}) \sqsupseteq S]$$

2.3 Trace Differential Privacy

In all of the previous definitions of differential privacy, the data set is determined before the automaton even starts interacting with the environment. In reality, often the data set is loaded into the automaton over time. In fact, it may even change between queries. Thus, we provide definitions for automata that accept their data set over time.

Now the inputs must represent not only queries but also the input of data points. Thus, we partition I into the set of queries Q and the set of data points D. These two sorts of inputs may be thought of as coming from two different access levels. The data points come from a trusted data provider who has access to the raw data and loads it into the system. The queries come from an untrusted data examiner or analyst who interacts with the system and gets access to the sanitized outputs.

Likewise, we partition the outputs into outputs for the trusted data provider and outputs for the untrusted data examiner. Since the outputs to the data provider is irrelevant to our definitions, we ignore them and let the set of outputs O range over the outputs to the data examiner. We let E range over all actions to which the examiner has direct access: $E = Q \cup O$.

The goal of differential privacy in this context is to prevent the data examiner from learning too much about any one data point provided to the system. This goal is similar to the security property noninterference [10]. However, by basing our definition on differential privacy, we avoid the overly conservative nature of noninterference.

Before we provide our definition for trace differential privacy, which takes into account the dynamic nature of providing inputs over time, we provide some formal notation.

Sequences. We use [ ] for the empty sequence. Let $a{:}\vec{a}$ be the sequence $\vec{a}$ with a prepended to it. Since we never deal with sequences of sequences, we abuse notation and use $\vec{a}{:}\vec{a}'$ for the sequence created by appending $\vec{a}'$ to $\vec{a}$ and $\vec{a}{:}a$ for $\vec{a}$ with $[a]$ appended to it.

We write $\lfloor \vec{a} \rfloor_S$ for restricting the action sequence $\vec{a}$ to some subset S of A. Formally, $\lfloor [\,] \rfloor_S = [\,]$ and

$$\lfloor a{:}\vec{a} \rfloor_S = \begin{cases} a{:}\lfloor \vec{a} \rfloor_S & a \in S \\ \lfloor \vec{a} \rfloor_S & \text{otherwise} \end{cases}$$

Given the input sequences $\vec{i}_1$ and $\vec{i}_2$, $\Delta(\vec{i}_1, \vec{i}_2)$ denotes the number of data points on which they differ. Formally,

- $\Delta(\vec{i}_1, \vec{i}_2) = 0$ iff $\vec{i}_1 = \vec{i}_2$.
- For $1 \leq n$, $\Delta(\vec{i}_1, \vec{i}_2) = n$ iff there exists $d \in D$, $\vec{i}, \vec{i}'_1, \vec{i}'_2 \in I^*$, such that both of the following properties hold:
  - either $\vec{i}_1 = \vec{i}{:}d{:}\vec{i}'_1$ and $\vec{i}_2 = \vec{i}{:}\vec{i}'_2$, or $\vec{i}_1 = \vec{i}{:}\vec{i}'_1$ and $\vec{i}_2 = \vec{i}{:}d{:}\vec{i}'_2$; and
  - $\Delta(\vec{i}'_1, \vec{i}'_2) = n - 1$.

Note that for $\Delta(\vec{i}_1, \vec{i}_2) = n$ to hold for any n, $\vec{i}_1$ and $\vec{i}_2$ must agree on every query from Q: they may only differ by n data points from D. Since differential privacy is essentially concerned with relative privacy of data sets differing on one element, in most theorems we consider the special case of the above definition for $n = 1$. That is, $\Delta(\vec{i}_1, \vec{i}_2) = 1$ iff there exists $d \in D$, and $\vec{i}, \vec{i}' \in I^*$ such that either $\vec{i}_1 = \vec{i}{:}d{:}\vec{i}'$ and $\vec{i}_2 = \vec{i}{:}\vec{i}'$, or $\vec{i}_2 = \vec{i}{:}d{:}\vec{i}'$ and $\vec{i}_1 = \vec{i}{:}\vec{i}'$.

Example. Let d_i range over elements in D and q_i range over elements in Q.

- $\Delta([d_1, q_1, d_2], [d_1, q_1]) = 1$ (one is a prefix of the other with the same queries).
- $\Delta([d_1, q_1, d_2, q_2], [q_1, d_2, q_2]) = 1$ (one is a suffix of the other with the same queries).
- $\Delta([d_1, q_1, d_2, q_2], [d_1, q_1, q_2]) = 1$ (the two sequences differ on one intermediate data point).
- $\Delta([d_1, d_2, q_1, q_2], [d_1, d_2, q_2, q_1])$ is undefined (the two sequences do not agree on queries).

The reason for forcing the order of queries to be the same in the definition of Δ is that in systems, the ordering of observable actions can be used as channels to compromise privacy. This is an example of an issue that arises when we move from abstract functions to concrete systems.

Definition 4 (Strong Trace Differential Privacy)**.** *A system K with queries Q, data points D, and outputs O gives ϵ-strong trace differential privacy if for all input sequences $\vec{i}_1$ and $\vec{i}_2$ in I^* such that $\Delta(\vec{i}_1, \vec{i}_2) \leq 1$ and for all $S \subseteq E^*$,*

$$\Pr[\lfloor K(\vec{i}_1) \rfloor_E \sqsupseteq S] \leq \exp(\epsilon) * \Pr[\lfloor K(\vec{i}_2) \rfloor_E \sqsupseteq S]$$

By restricting $K(\vec{i}_1)$ to only those elements of $E = Q \cup O$, we limit traces to only those actions accessible to the untrusted data examiner. The definition requires that any subset of such traces be almost equally probable under the input sequences $\vec{i}_1$ and $\vec{i}_2$, which differ by at most one data point. The order of $\vec{i}_1$ and $\vec{i}_2$ matters since the answers might only make sense for some queries. Thus, an appropriate query must precede each answer.

A weak version of trace differential privacy is also possible. Here we take c to be function over queries in Q to the corresponding ϵ_q and raise c to work over input sequences as follows: $c([]) = 0$, $c(q{:}\vec{i}) = c(q) + c(\vec{i})$ for $q \in Q$, and $c(d{:}\vec{i}) = c(\vec{i})$ for $d \in D$.

Definition 5 (Weak Trace Differential Privacy)**.** *A system K with queries Q, data points D, and outputs O gives c-weak trace differential privacy if for all input sequences $\vec{i}_1$ and $\vec{i}_2$ in I^* such that $\Delta(\vec{i}_1, \vec{i}_2) \leq 1$ and for all $S \subseteq E^*$,*

$$\Pr[\lfloor K(\vec{i}_1) \rfloor_E \sqsupseteq S] \leq \exp(c(\vec{i}_1)) * \Pr[\lfloor K(\vec{i}_2) \rfloor_E \sqsupseteq S]$$

Note the choice of $c(\vec{i}_1)$ could be replaced by $c(\vec{i}_2)$ since $c(\vec{i}_1) = c(\vec{i}_2)$ for all $\vec{i}_1$ and $\vec{i}_2$ such that $\Delta(\vec{i}_1, \vec{i}_2) \leq 1$.

2.4 Properties of Trace Differential Privacy

In this section we present some basic theorems about trace differential privacy. Although the definitions of trace differential privacy refer to all possible subsets of the set of traces, Theorems 1-2 prove that they are equivalent to definitions that refer to each trace separately. This result simplifies reasoning about trace differential privacy and is useful for proving subsequent results in this paper. Theorems 3-4 are composition theorems that are analogous to previous results about differential privacy for functions: they prove that the privacy error bound for a system whose inputs differ on at most n data points is $n * \epsilon$, where ϵ is the error bound for the system if its inputs differ on one data point.

From sets to single sequences. Since we have restricted ourselves to a countable set of actions A, we need only reason about one possible sequence $\vec{e}$ of E^* at a time, not about all possible subsets of E^* as required by Definition 4. The following theorem formalizes this observation. We use it in the proof of other theorems, for example, Theorem 8.

Theorem 1. *K has ϵ-strong trace differential privacy if and only if for all input sequences $\vec{i}_1$ and $\vec{i}_2$ in I such that $\Delta(\vec{i}_1, \vec{i}_2) \leq 1$ and $\vec{e}$ in E^*,*

$$\Pr[\lfloor K(\vec{i}_1)\rfloor_E \sqsupseteq \vec{e}] \leq \exp(\epsilon) * \Pr[\lfloor K(\vec{i}_2)\rfloor_E \sqsupseteq \vec{e}]$$

Proof. The only if direction follows directly from the definition by setting $S = \{\vec{e}\}$.

For the if direction, arbitrarily fix $\vec{i}_1$ and $\vec{i}_2$ such that $\Delta(\vec{i}_1, \vec{i}_2) \leq 1$ and $S \subseteq E^*$. By assumption, for all $\vec{e}$ in E^*,

$$\Pr[\lfloor K(\vec{i}_1)\rfloor_E \sqsupseteq \vec{e}] \leq \exp(\epsilon) * \Pr[\lfloor K(\vec{i}_2)\rfloor_E \sqsupseteq \vec{e}]$$

Let S' be S with all the elements that are a longer version of another element of S removed. That is, $S' = \{\, \vec{e}' \in S \mid \nexists \vec{e} \in S \,\text{s.t.}\, \vec{e}' \sqsupset \vec{e} \}$ where $\vec{e}' \sqsupset \vec{e}$ means that $\vec{e}$ is a strict prefix of $\vec{e}'$. Proof by induction over the length of $\vec{e}$ shows that for all $\vec{e}$ in S, there exists $\vec{e}'$ in S' such that $\vec{e} \sqsupseteq \vec{e}'$. Thus, if there exists $\vec{e}$ in S such that $\lfloor K(\vec{i}_1)\rfloor_E \sqsupseteq \vec{e}$, then there exists $\vec{e}'$ in S' such that $\lfloor K(\vec{i}_1)\rfloor_E \sqsupseteq \vec{e}'$. Thus, for all $\vec{i}$, $\Pr[\lfloor K(\vec{i})\rfloor_E \sqsupseteq S] = \Pr[\lfloor K(\vec{i})\rfloor_E \sqsupseteq S']$.

For two $\vec{e}'_1$ and $\vec{e}'_2$ in S' such that $\vec{e}'_1 \neq \vec{e}'_2$, $\lfloor K(\vec{i})\rfloor_E$ can only have one of them as a prefix since neither is a prefix of the other. Thus, since S is countable, this implies that

$$\Pr[\lfloor K(\vec{i})\rfloor_E \sqsupseteq S] = \sum_{\vec{e}' \in S'} \Pr[\lfloor K(\vec{i})\rfloor_E \sqsupseteq \vec{e}']$$

Thus,

$$\begin{aligned}
\Pr[\lfloor K(\vec{i}_1)\rfloor_E \sqsupseteq S] &= \Pr[\lfloor K(\vec{i}_1)\rfloor_E \sqsupseteq S'] \\
&= \sum_{\vec{e}' \in S'} \Pr[\lfloor K(\vec{i}_1)\rfloor_E \sqsupseteq \vec{e}'] \\
&\leq \sum_{\vec{e}' \in S'} \exp(\epsilon) * \Pr[\lfloor K(\vec{i}_2)\rfloor_E \sqsupseteq \vec{e}'] \\
&= \exp(\epsilon) \sum_{\vec{e}' \in S'} \Pr[\lfloor K(\vec{i}_2)\rfloor_E \sqsupseteq \vec{e}'] \\
&= \exp(\epsilon) \Pr[\lfloor K(\vec{i}_2)\rfloor_E \sqsupseteq S'] \\
&= \exp(\epsilon) \Pr[\lfloor K(\vec{i}_2)\rfloor_E \sqsupseteq S]
\end{aligned}$$

□

Theorem 2. *K has c-weak trace differential privacy if and only if for all input sequences $\vec{i}_1$ and $\vec{i}_2$ in I^* such that $\Delta(\vec{i}_1, \vec{i}_2) \leq 1$ and $\vec{e}$ in E^*,*

$$\Pr[\lfloor K(\vec{i}_1)\rfloor_E = \vec{e}] \leq \exp(c(\vec{i}_1)) * \Pr[\lfloor K(\vec{i}_2)\rfloor_E = \vec{e}]$$

Proof. This proof is identical to the proof of Theorem 1 substituting $c(\vec{i}_1)$ for ϵ. □

Data sets differing on n elements. Differential privacy has a compositionality property that states the outputs of function κ with ϵ-differential privacy is related by $n * \epsilon$ on two data sets that differ by n elements. We prove an analogous theorem.

Theorem 3. *If an automaton K has ϵ-strong trace differential privacy, then for all inputs sequences $\vec{i}_1$ and $\vec{i}_2$ such that $\Delta(\vec{i}_1, \vec{i}_2) \leq n$ and for all $S \subseteq E^*$,*

$$\Pr[\lfloor K(\vec{i}_1)\rfloor_E \sqsupseteq S] \leq \exp(n * \epsilon) * \Pr[\lfloor K(\vec{i}_2)\rfloor_E \sqsupseteq S]$$

The proof of this theorem is a simpler version of the proof for the weak version given below.

Theorem 4. *If an automaton K has c-weak trace differential privacy, then for all inputs sequences $\vec{i}_1$ and $\vec{i}_2$ such that $\Delta(\vec{i}_1, \vec{i}_2) \leq n$ and for all $S \subseteq E^*$,*

$$\Pr[\lfloor K(\vec{i}_1) \rfloor_E \sqsupseteq S] \leq \exp(n * c(\vec{i}_1)) * \Pr[\lfloor K(\vec{i}_2) \rfloor_E \sqsupseteq S]$$

Proof. Proof by induction over n.

Base Case: $n = 0$. In this case, $\vec{i}_1 = \vec{i}_2$ and, thus, $\Pr[\lfloor K(\vec{i}_1) \rfloor_E \sqsupseteq S] = \Pr[\lfloor K(\vec{i}_2) \rfloor_E \sqsupseteq S]$ as needed with $\exp(0) = 1$.

Inductive Case: Assume for all $n \leq m$; prove for $m + 1$. Since $\Delta(\vec{i}_1, \vec{i}_2) = m + 1$, there must exist $\vec{i}, \vec{i}'_1, \vec{i}'_2 \in I^*$ and $d_1 \in D$ such that $\vec{i}_1 = \vec{i}{:}d_1{:}\vec{i}'_1$, $\vec{i}_2 = \vec{i}{:}\vec{i}'_2$, and $\Delta(\vec{i}'_1, \vec{i}'_2) = m$. Let $\vec{i}_3 = \vec{i}{:}i_2{:}\vec{i}'_1$. $c(\vec{i}_1) = c(\vec{i}_3)$, $\Delta(\vec{i}_1, \vec{i}_3) = 1$, and $\Delta(\vec{i}_3, \vec{i}_2) = m$. Thus, by the inductive hypothesis,

$$\Pr[\lfloor K(\vec{i}_1) \rfloor_E \sqsupseteq S] \leq \exp(1 * c(\vec{i}_1)) * \Pr[\lfloor K(\vec{i}_3) \rfloor_E \sqsupseteq S]$$

and

$$\Pr[\lfloor K(\vec{i}_3) \rfloor_E \sqsupseteq S] \leq \exp(m * c(\vec{i}_3)) * \Pr[\lfloor K(\vec{i}_2) \rfloor_E \sqsupseteq S]$$

Thus,

$$\begin{aligned}\Pr[\lfloor K(\vec{i}_1) \rfloor_E \sqsupseteq S] &\leq \exp(1 * c(\vec{i}_1)) * \Big(\exp(m * c(\vec{i}_3)) * \Pr[\lfloor K(\vec{i}_2) \rfloor_E \sqsupseteq S]\Big) \\ &= \exp((m+1) * c(\vec{i}_1)) * \Pr[\lfloor K(\vec{i}_2) \rfloor_E \sqsupseteq S]\end{aligned}$$

as needed. □

3 Relating the Definitions

In this section, we prove several theorems (Theorems 5-8) characterizing the relationship between the original definition of differential privacy for functions and our definitions of trace differential privacy. Section 3.1 presents results that indicate that trace differential privacy implies the original notion of differential privacy for functions under certain conditions. Section 3.2 presents results for constructing a system that satisfies trace differential privacy from functions that individually satisfy differential privacy. Such results are useful because they allow us to use existing constructions for differentially private functions to achieve trace differential privacy. We prove that a precisely defined class of automata that are *synchronous database implementations* of a set of functions satisfy weak trace differential privacy. In addition, we demonstrate via examples that asynchrony in real systems can cause trace differential privacy to be violated even if we use differentially private functions to respond to queries. Finally, in Section 3.3, we present a result that enables trace differential privacy guarantees with tight privacy error bounds to be established for data sets that change over time.

3.1 From Systems to Functions

Now we justify calling our definitions notions of differential privacy. Intuitively, if an automaton K has weak trace differential privacy, then K treated as a function on data sets will have differential privacy. To treat K as a function and to avoid problems in the original definition of differential privacy involving non-measurable sets, we assume that K is terminating and that termination is noticeable. That is, we assume that for all $\vec{i}$, $K(\vec{i})$ is an element of A^* and ends in a distinguished action done in O that can only show up at the end of traces. Whereas the set of observable traces was before E^*, we now limit them to be $E^*_{\mathsf{done}} = \{\, \vec{e} \in E^* \mid \exists \vec{e}' \in E^* \text{ s.t. } \vec{e} = \vec{e}'{:}\mathsf{done} \,\}$. In this setting $\Pr[\lfloor K(\vec{i}) \rfloor_E \in S]$ is well defined for subsets S of E^*_{done}. Furthermore, by making termination noticeable, observing a sequences of actions ending in done allows one to conclude that the sequence of actions is not merely a prefix of the trace but the whole trace

that the system will produce under the given inputs. As show in the proof of Theorem 5, this property is key to providing tight privacy error bounds.

Since K, unlike a function over data sets, accepts both data points and queries, we must fix a single query q that the function treatment of K will answer. Furthermore, since the the result of evaluating the function on a given set of data points will depend on the behavior of K, which accepts sequences of data points, we must fix a way to convert a set into a sequence. Given a data set B, which is a multiset with elements from D, and total ordering $\leq$ on the data points in D, let $\mathsf{toSeq}(B, \leq)$ be the sequence of data points that results from sorting B by $\leq$. We let $\kappa(K, q, \leq)$ represent K treated as function over data sets ordered by $\leq$ and answering the query q. Formally, $\kappa(K, q, \leq)(B) = \lfloor K(\mathsf{toSeq}(B, \leq){:}q) \rfloor_E$. Assuming that K terminates, $\kappa(K, q, \leq)$ is a probabilistic function with the range E^*_{done}.

Theorem 5. *If a noticeably terminating automaton K has c-weak trace differential privacy and terminates, then for all q in Q and total orderings $\leq$ of D, $\kappa(K, q, \leq)$ has $c(q)$-differential privacy.*

Proof. Arbitrarily fix $q \in Q$, $S \subseteq E^*_{\mathsf{done}}$, and a total ordering $\leq$. Let B_1 and B_2 be two multisets with elements from D such that they differ by at most one element. Let $\vec{d_1} = \mathsf{toSeq}(B_1, \leq)$ and $\vec{d_2} = \mathsf{toSeq}(B_2, \leq)$.

Since B_1 and B_2 differ by at most one element and $\vec{d_1}$ and $\vec{d_2}$ are constructed using the same ordering, $\Delta(\vec{d_1}, \vec{d_2}) \leq 1$. Thus, since K has c-weak trace differential privacy,

$$\Pr[\lfloor K(\vec{d_1}{:}q) \rfloor_E \sqsupseteq S] \leq \exp(c(q)) * \Pr[\lfloor K(\vec{d_2}{:}q) \rfloor_E \sqsupseteq S]$$

since $c(\vec{d_1}{:}q) = c(q)$.

In general, it is possible that $\lfloor K(\vec{d_1}{:}q) \rfloor_E \sqsupseteq S$ but not $\lfloor K(\vec{d_1}{:}q) \rfloor_E \in S$. This can happen when there exists $\vec{e}$ in S such that $\lfloor K(\vec{d_1}{:}q) \rfloor_E$ has $\vec{e}$ in S as a prefix, but $\lfloor K(\vec{d_1}{:}q) \rfloor_E$ is not equal to $\vec{e}$ by being longer than $\vec{e}$. However, since every $\vec{e} \in S$ ends with done, we know that $\lfloor K(\vec{d_1}{:}q) \rfloor_E$ can only have $\vec{e}$ as a prefix if it includes done. Since done cannot come before the end of an observation, this means that $\lfloor K(\vec{d_1}{:}q) \rfloor_E$ cannot be longer than $\vec{e}$ and must be equal to it. Thus, $\lfloor K(\vec{d_1}{:}q) \rfloor_E \sqsupseteq S$ if and only if $\lfloor K(\vec{d_1}{:}q) \rfloor_E \in S$.

Since $\lfloor K(\vec{d_1}{:}q) \rfloor_E = \kappa(K, q, \leq)(B_1)$ and $\lfloor K(\vec{d_2}{:}q) \rfloor_E = \kappa(K, q, \leq)(B_2)$. Thus,

$$\Pr[\kappa(K, q, \leq)(B_1) \in S] \leq \exp(c(q)) * \Pr[\kappa(K, q, \leq)(B_2) \in S]$$

□

To avoid the requirements of noticeable termination, we could change the definition of differential privacy to only look at prefixes by replacing $\in$ with $\sqsupseteq$ in Definition 1. We refer to this variant of differential privacy as *prefix differential privacy.*

Theorem 6. *If an automaton K has c-weak trace differential privacy, then for all q in Q and total orderings $\leq$ of D, $\kappa(K, q, \leq)$ has $c(q)$-prefix differential privacy.*

Proof. Arbitrarily fix $q \in Q$, $S \subseteq E^*$, and a total ordering $\leq$. Let B_1 and B_2 be two multisets with elements from D such that they differ by at most one element. Let $\vec{d_1} = \mathsf{toSeq}(B_1, \leq)$ and $\vec{d_2} = \mathsf{toSeq}(B_2, \leq)$.

Since B_1 and B_2 differ by at most one element and $\vec{d_1}$ and $\vec{d_2}$ are constructed using the same ordering, $\Delta(\vec{d_1}, \vec{d_2}) \leq 1$. Thus, since K has c-weak trace differential privacy,

$$\Pr[\lfloor K(\vec{d_1}{:}q) \rfloor_E \sqsupseteq S] \leq \exp(c(q)) * \Pr[\lfloor K(\vec{d_2}{:}q) \rfloor_E \sqsupseteq S]$$

since $c(\vec{d_1}{:}q) = c(q)$. $\lfloor K(\vec{d_1}{:}q) \rfloor_E = \kappa(K, q, \leq)(B_1)$ and $\lfloor K(\vec{d_2}{:}q) \rfloor_E = \kappa(K, q, \leq)(B_2)$. Thus,

$$\Pr[\kappa(K, q, \leq)(B_1) \in S] \leq \exp(c(q)) * \Pr[\kappa(K, q, \leq)(B_2) \in S]$$

□

Note that this does not imply that for two different orderings $\leq_1$ and $\leq_2$,

$$\Pr[\kappa(K, q, \leq_1)(B_1) \sqsupseteq S] \leq \exp(\epsilon) * \Pr[\kappa(K, q, \leq_2)(B_2) \sqsupseteq S]$$

For such a result, $\Delta(\vec{i_1}, \vec{i_2}) \leq 1$ would have to hold when $\vec{i_1}$ is any permutation of $\vec{i_2}$. This would produce too strong of a notion of differential privacy since the decision as to whether or not to add oneself to a study results only in the addition of a input, not a permutation of the input.

3.2 From Functions to Systems

It is also possible to construct a differentially private system from a set of differentially private functions. For example, consider a system that answers the queries by applying a differentially private function to the received inputs treating them as a set. To formalize this idea, let $\mathsf{toMultiset}(\vec{i})$ represent the multiset of all elements in $\vec{i}$. Let $\varkappa$ be a set functions and c be a mapping such that for all $\kappa \in \varkappa$, κ has $c(\kappa)$-differential privacy. Furthermore, assume that all κ_1 and κ_2 in $\varkappa$ such that $\kappa_1 \neq \kappa_2$ are independent: for all multisets B_1 and B_2 with elements from D and $o_1, o_2 \in O$, $\Pr[\kappa_1(D_1) = o_1 \wedge \kappa_2(D_2) = o_2] = \Pr[\kappa_1(D_1) = o_1] * \Pr[\kappa_2(D_2) = o_2]$ where we treat $\Pr[\kappa(D) = o]$ as 0 when o is not in the range of κ.

Suppose K is a system such that its queries Q are indexes into $\varkappa$ where for each q, κ_q represents the sanitization function of $\varkappa$ indexed by q. The system will not accept a new query as input until it has produced a response to the previous query as output. Let D be those data points acceptable to all the functions in $\varkappa$. Let O be the union of the ranges of all functions in $\varkappa$. Let $\eta(\vec{i}, j)$ denote all the data points in $\vec{i}$ before the jth query. That is, $\eta(\vec{i}, 0) = [\,]$, $\eta(d{:}\vec{i}, j) = d{:}\eta(\vec{i}, j)$, and $\eta(q{:}\vec{i}, j) = \eta(\vec{i}, j-1)$ for all $j > 0$. Let $\vec{a}[j]$ be the jth element of the sequence $\vec{a}$. Formally, K is a *synchronous database implementation* of $\varkappa$ if the following property holds: $\Pr[\lfloor K(\vec{i})\rfloor_E = \vec{e}] = \prod_{j=1}^{|\vec{e}|/2} \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i}, 2j-1))) = \vec{e}[2j]]$ where $\lfloor \vec{i}\rfloor_Q = \lfloor \vec{e}\rfloor_Q$, $|\vec{e}| = 2 * |\lfloor \vec{i}\rfloor_Q|$, and for all j from 1 to $|\vec{e}|/2$, $\vec{e}[2j-1] \in Q$ and $\vec{e}[2j] \in O$, and $\Pr[\lfloor K(\vec{i})\rfloor_E = \vec{e}] = 0$ otherwise. Note here that the indices in the definition are into sequences obtained by restricting the action sequences to queries and outputs, and do not constrain the data entries in between.

Theorem 7. *For all $\varkappa$ and c such that all $\kappa \in \varkappa$ has $c(\kappa)$-differential privacy, any K that is a synchronous database implementation of $\varkappa$ has c-weak trace differential privacy.*

Proof. Arbitrarily fix $\vec{i_1}$ and $\vec{i_2}$ such that $\Delta(\vec{i_1}, \vec{i_2}) \leq 1$.

Note that for all $\vec{e}$ if $|\vec{e}| \geq 2 * |\lfloor \vec{i_1}\rfloor_Q| = 2 * |\lfloor \vec{i_2}\rfloor_Q|$, then $\Pr[\lfloor K(\vec{i_1})\rfloor_E \sqsupseteq \vec{e}] = 0 = \Pr[\lfloor K(\vec{i_2})\rfloor_E \sqsupseteq \vec{e}]$.

We prove that for all $\vec{e}$ such that $|\vec{e}| \leq 2 * |\lfloor \vec{i_1}\rfloor_Q|$, $\Pr[\lfloor K(\vec{i_1})\rfloor_E \sqsupseteq \vec{e}] \leq \exp(c(\vec{i_1}))\Pr[\lfloor K(\vec{i_2})\rfloor_E \sqsupseteq \vec{e}]$ by induction over the length of $|\vec{e}|$. However, rather than starting at zero and working up, we start with $|\vec{e}| = 2 * |\lfloor \vec{i_1}\rfloor_Q|$ and work our way down.

- Base Case: $|\vec{e}| = 2 * |\lfloor \vec{i_1}\rfloor_Q|$.

  In the subcase where for all j from 1 to $|\vec{e}|/2$, $\vec{e}[2j-1] \in Q$ and $\vec{e}[2j] \in O$,

  $$\Pr[\lfloor K(\vec{i})\rfloor_E = \vec{e}] = \prod_{j=1}^{|\vec{e}|/2} \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i}, 2j-1))) = \vec{e}[2j]]$$

  Since $\vec{i_1}$ and $\vec{i_2}$ differ by at most one data point, $\eta(\vec{i_1}, 2j-1)$ and $\eta(\vec{i_2}, 2j-1)$ will differ by at most data point. Thus, the difference between $\mathsf{toMultiset}(\eta(\vec{i_1}, 2j-1))$ and $\mathsf{toMultiset}(\eta(\vec{i_2}, 2j-1))$ is at most one. Since for all $\kappa \in \varkappa$, κ has $c(\kappa)$-differential privacy and they are all independent, for all j from 1 to $|\vec{e}|/2$, $\Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i_1}, 2j-1))) = \vec{e}[2j]] \leq \exp(c(\kappa_{\vec{e}[2j-1]}))\Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i_2}, 2j-1))) =$

$\vec{e}[2j]]$. Thus,

$$\begin{aligned}
&\Pr[\lfloor K(\vec{i})\rfloor_E = \vec{e}] \\
&\quad= \prod_{j=1}^{|\vec{e}|/2} \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i}_1, 2j-1))) = \vec{e}[2j]] \\
&\quad\leq \prod_{j=1}^{|\vec{e}|/2} \exp(c(\vec{e}[2j-1])) \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i}_2, j))) = \vec{e}[2j]] \\
&\quad= \left(\prod_{j=1}^{|\vec{e}|/2} \exp(c(\vec{e}[2j-1]))\right) \left(\prod_{j=1}^{|\vec{e}|/2} \exp(c(\vec{e}[2j-1])) \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i}_2, j))) = \vec{e}[2j]]\right) \\
&\quad= \exp\left(\sum_{j=1}^{|\vec{e}|/2} c(\vec{e}[2j-1])\right) \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i}_2, j))) = \vec{e}[2j]] \\
&\quad= \exp(c(\vec{i}_1)) \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i}_2, j))) = \vec{e}[2j]]
\end{aligned}$$

Where the last line follows from $\sum_{j=1}^{|\vec{e}|/2} c(\vec{e}[2j-1]) = \sum_{j=1}^{|\lfloor\vec{e}\rfloor_I|} c(\lfloor\vec{e}\rfloor_I[j]) = \sum_{j=1}^{|\lfloor\vec{i}_1\rfloor_Q|} c(\lfloor\vec{i}_1\rfloor_Q[j]) = c(\vec{i}_1)$.

In the other subcase, where $|\vec{e}|$ is not even or for some j from 0 to $|\vec{e}|/2$, $\vec{e}[2j-1] \notin I$ or $\vec{e}[2j] \notin O$, $\Pr[\lfloor K(\vec{i}_1)\rfloor_Q = \vec{e}] = 0 = \Pr[\lfloor K(\vec{i}_2)\rfloor_Q = \vec{e}]$.

Thus, in either subcase, $\Pr[\lfloor K(\vec{i}_1)\rfloor_Q = \vec{e}] \leq \exp(c(\vec{i})) \Pr[\lfloor K(\vec{i}_2)\rfloor_Q = \vec{e}]$. $\Pr[\lfloor K(\vec{i}_1)\rfloor_Q \sqsupseteq \vec{e}'] = 0$ for all $\vec{e}'$ that extend $\vec{e}$, $\Pr[\lfloor K(\vec{i}_1)\rfloor_Q \sqsupseteq \vec{e}'] = \Pr[\lfloor K(\vec{i}_1)\rfloor_Q = \vec{e}']$. Similarly, $\Pr[\lfloor K(\vec{i}_2)\rfloor_Q \sqsupseteq \vec{e}'] = \Pr[\lfloor K(\vec{i}_2)\rfloor_Q = \vec{e}']$. Thus, $\Pr[\lfloor K(\vec{i}_1)\rfloor_Q \sqsupseteq \vec{e}] \leq \exp(c(\vec{i})) \Pr[\lfloor K(\vec{i}_2)\rfloor_Q \sqsupseteq \vec{e}]$.

- Inductive Case: Assume for all $|\vec{e}'| \geq n$; prove for $|\vec{e}| = n-1$. $\Pr[\lfloor K(\vec{i}_1)\rfloor_Q \sqsupseteq \vec{e}] = \Pr[\lfloor K(\vec{i}_1)\rfloor_Q = \vec{e} \vee \bigvee_{e\in E} \lfloor K(\vec{i}_1)\rfloor_Q \sqsupseteq \vec{e}{:}e]$. Since $|\vec{e}| < 2 * |\lfloor\vec{i}_1\rfloor_Q|$, $\Pr[\lfloor K(\vec{i}_1)\rfloor_Q = \vec{e}] = 0$ and each event under the disjunction is mutually exclusive, $\Pr[\lfloor K(\vec{i}_1)\rfloor_Q \sqsupseteq \vec{e}] = \sum_{e\in E} \Pr[\lfloor K(\vec{i}_1)\rfloor_Q \sqsupseteq \vec{e}{:}e]$. Similarly, $\Pr[\lfloor K(\vec{i}_2)\rfloor_Q \sqsupseteq \vec{e}] = \sum_{e\in E} \Pr[\lfloor K(\vec{i}_2)\rfloor_Q \sqsupseteq \vec{e}{:}e]$. Since each $\vec{e}{:}e$ is longer, we can apply the inductive hypothesis and get that

$$\begin{aligned}
Pr[\lfloor K(\vec{i}_1)\rfloor_Q \sqsupseteq \vec{e}] &= \sum_{e\in E} \Pr[\lfloor K(\vec{i}_1)\rfloor_Q \sqsupseteq \vec{e}{:}e] \\
&\leq \sum_{e\in E} \exp(c(\vec{i}_1)) \Pr[\lfloor K(\vec{i}_2)\rfloor_Q \sqsupseteq \vec{e}{:}e] \\
&= \exp(c(\vec{i}_1)) \Pr[\lfloor K(\vec{i}_2)\rfloor_Q \sqsupseteq \vec{e}]
\end{aligned}$$

Finally, we use Theorem 2 to prove c-weak differential privacy. □

Asynchrony and scheduling. The above theorem is closely related "sequential composition" of differentially private functions, which was used by Dwork [7] and stated by McSherry and Talwar [18] as

> The sequential application of mechanism $\{M_i\}$, each giving $\{\epsilon_i\}$-differential privacy, gives $(\sum_i \epsilon_i)$-differential privacy.

and which is made more formal by McSherry [17, 3]. Both our work and theirs consider a set of differentially private functions and model the outputs as a sequence of answers in a fixed order. Likewise, in both cases the privacy provided is a sum based on the number of queries asked and the privacy of each query. In this sense, our model preserves some existing results about "synchronous" implementations. However, it should be noted that our model and definitions are general enough to move beyond such systems and reason about differential privacy of asynchronous systems.

Some subtle issues about scheduling should be dealt with care in obtaining similar results for asynchronous systems. Asynchronous systems can both answer every query in a differentially private manner and still not be differentially private by changing the order in which queries are answered, or by correlating the choices made by the scheduler and the differentially private sanitization functions.

As a simple example, consider a system that after consuming a distinguished query $q^\dagger \in Q$ waits until it receives a second query before answering it and answers the two queries in an order determined by first data point received. Here, the problem arises because the choices of the scheduler (part of the system that decides which query to answer) may depend on the past data entries.

As a more subtle example, consider a system K such that the scheduler's choices may not depend on the past data points. Suppose there are two possible queries q_1 and q_2 (with corresponding differentially private functions κ_1 and κ_2 respectively), and the scheduler that always chooses to answer q_1 or q_2 with the equal probability 1/2, without depending on the past data entries. Suppose κ_1 is a function that outputs o_1^1 or o_1^2, each with probability 1/2 and κ_2 is a function that outputs o_2 with probability 1, regardless of past data entries. Now, consider two input sequences $\vec{i}_1 = [q_1, q_2]$ and $\vec{i}_2 = [d, q_1, q_2]$ where $d \in D$. K can behave so that if the scheduler chooses to answer q_1 first,

- For i_1, the probability of queries being followed by $[o_1^1, o_2]$ is 1/2 and the probability of queries being followed by $[o_1^2, o_2]$ is 0, and
- For i_2 the probability of queries being followed by $[o_1^2, o_2]$ is 1/2 and the probability of queries being followed by $[o_1^1, o_2]$ is 0.

Similarly, if the scheduler chooses to answer q_2 first,

- For i_1, the probability of queries being followed by $[o_2, o_1^2]$is 1/2 and the probability of queries being followed by $[o_2, o_1^1]$ is 0, and
- For i_2, the probability of queries being followed by $[o_2, o_1^1]$ is 1/2 and the probability of queries being followed by $[o_2, o_1^2]$ is 0.

That is, even though the scheduler component and query sanitization functions do not depend on data points, there can be a dependence between the scheduler component and the functions that causes a violation of privacy caught by our definition of trace differential privacy.

An in-depth investigation of interaction between scheduling and differential privacy is interesting in its own right but is outside of the scope of this paper.

3.3 Mutable Data Sets

One key difference of our definitions from the original definition of differential privacy is the ability to talk about changing data sets. Consider the question of constructing a system that maintains ϵ-strong trace differential privacy. One way to maintain ϵ-strong trace differential privacy would be to provide a system such as a synchronous database implementation of $\varkappa$ except it refuses to answer any more queries if answering another would make $c(\vec{i})$ larger than the ϵ bound. However, such a system would be too conservative. The ability to address changing data sets offers us a less conservative approach to maintaining ϵ-strong trace differential privacy.

Some of the queries contributing to $c(\vec{i})$ may have occurred before some of the data points entered the data set. It would be safe to answer additional queries provided that they only use data points not involved in all the queries. This system effectively removes old points from the data set.

For example, consider the action sequence

$$\vec{a} = [d_1, d_2, d_3, q_1, o_1, d_4, d_5, q_2, o_2, d_6, d_7, d_8, q_3, o_3, d_9, q_4, o_4]$$

Suppose that q_1, q_2, q_3, and q_4 are queries that correspond to sanitization functions κ_{q_1}, κ_{q_2}, κ_{q_3}, and κ_{q_4} each of which provide ϵ-differential privacy. In a simple system that simply answers each query based on the

data points that precede it, like the synchronous database in Section 3.2, the privacy error bound for the whole computation would be $4 * \epsilon$ since 4 queries are answered. However, suppose a privacy error bound of no more than $2 * \epsilon$ is considered acceptable. One way to achieve this would be to stop using data points after the system provides them as inputs to two sanitization functions. Under this approach, the data points d_1, d_2, and d_3 would not be provided as input to κ_{q_3} in calculating o_3 since they were already used to calculate o_1 and o_2. One may visualize this approach as a sliding window extending from an output back to the earliest data points on which that output may depend:

$$[\boxed{d_1, d_2, d_3, q_1, o_1}, d_4, d_5, q_2, o_2, d_6, d_7, d_8, q_3, o_3, d_9, q_4, o_4]$$
$$[\boxed{d_1, d_2, d_3, q_1, o_1, d_4, d_5, q_2, o_2}, d_6, d_7, d_8, q_3, o_3, d_9, q_4, o_4]$$
$$[d_1, d_2, d_3, q_1, o_1, \boxed{d_4, d_5, q_2, o_2, d_6, d_7, d_8, q_3, o_3}, d_9, q_4, o_4]$$
$$[d_1, d_2, d_3, q_1, o_1, d_4, d_5, q_2, o_2, \boxed{d_6, d_7, d_8, q_3, o_3, d_9, q_4, o_4}]$$

One may view this sliding window as holding all the data points in a mutable data set at the time that the output at the right end of the sliding window is answered. Data points before the sliding window corresponds to removed data points.

We formalize such a system in which for all $\kappa \in \varkappa$, $c(\kappa) = \epsilon$. Let $\eta'(\vec{i}, j', j)$ be those data points between the j'th query and the jth query. That is,

$$\begin{aligned}
\eta'(\vec{i}, j', 0) &= [] \\
\eta'(d{:}\vec{i}, j', j) &= \eta'(\vec{i}, j', j) && \text{where } j > 0 \text{ and } j' > 0 \\
\eta'(d{:}\vec{i}, j', j) &= d{:}\eta'(\vec{i}, j', j) && \text{where } j > 0 \text{ and } j' \leq 0 \\
\eta'(q{:}\vec{i}, j', j) &= \eta'(\vec{i}, j' - 1, j - 1) && \text{where } j > 0
\end{aligned}$$

Let use call a system a synchronous database implementation of $\varkappa$ individually capped at $t * \epsilon$ if $\Pr[\lfloor K(\vec{i}) \rfloor_Q = \vec{e}] = \prod_{j=1}^{|\vec{e}|/2} \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta'(\vec{i}_1, j - t, j))) = \vec{e}[2j]]$ where $\lfloor \vec{i} \rfloor_Q = \lfloor \vec{e} \rfloor_Q$, $|\vec{e}| = 2 * \lfloor \vec{i} \rfloor_Q$, and for all j from 0 to $|\vec{e}|/2$, $\vec{e}[2j-1] \in I$ and $\vec{e}[2j] \in O$; and $\Pr[\lfloor K(\vec{i}) \rfloor_E = \vec{e}] = 0$ otherwise.

Theorem 8. *For all $\varkappa$ such that all $\kappa \in \varkappa$ has ϵ-differential privacy, any K that is a synchronous database implementation of $\varkappa$ individually capped at $t * \epsilon$ has $(t * \epsilon)$-strong trace differential privacy.*

Proof. This proof proceeds mostly as the proof of Theorem 7 did. However, instead of proving that for all $\vec{e}$, $\Pr[\lfloor K(\vec{i}_1) \rfloor_Q \sqsupseteq \vec{e}] \leq \exp(c(\vec{i}_1)) Pr[\lfloor K(\vec{i}_2) \rfloor_Q \sqsupseteq \vec{e}]$, we instead prove that $\Pr[\lfloor K(\vec{i}_1) \rfloor_Q \sqsupseteq \vec{e}] \leq \exp(t * \epsilon) Pr[\lfloor K(\vec{i}_2) \rfloor_Q \sqsupseteq \vec{e}]$. By Theorem 1, this result implies $(t * \epsilon)$-strong differential privacy. The only major difference in the proof is in the base case of the induction, which is presented below.

Since $\vec{i}_1$ and $\vec{i}_2$ differ by at most one data point, $\eta'(\vec{i}_1, j - t, j)$ and $\eta'(\vec{i}_2, j - t, j)$ will differ by at most data point. Thus, the difference between $\mathsf{toMultiset}(\eta'(\vec{i}_1, j - t, j))$ and $\mathsf{toMultiset}(\eta(\vec{i}_2, j - t, j))$ is at most one. Furthermore, if d is not between $j - t$ and j, $\mathsf{toMultiset}(\eta'(\vec{i}_1, j - t, j)) = \mathsf{toMultiset}(\eta(\vec{i}_2, j - t, j))$. Only for at most t values of j will d be between $j - t$ and j. Let J denote these t values that are less than $|\vec{e}|/2$.

Since all $\kappa \in \varkappa$ are ϵ-differentially private and they are independent, for all $j \in J$,

$$\Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta'(\vec{i}_1, j - t, j))) = \vec{e}[2j]] \leq \exp(\epsilon) \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta'(\vec{i}_2, j - t, j))) = \vec{e}[2j]]$$

For all $j \notin J$ from 0 to $|\vec{e}|/2$, $\mathsf{toMultiset}(\eta'(\vec{i}_1, j-t, j)) = \mathsf{toMultiset}(\eta(\vec{i}_2, j-t, j))$. Thus,

$$\begin{aligned}
\Pr[\lfloor K(\vec{i})\rfloor_Q \sqsupseteq \vec{e}] &= \Pr[\lfloor K(\vec{i})\rfloor_Q = \vec{e}] \\
&= \prod_{j=1}^{|\vec{e}|/2} \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i}_1, j))) = \vec{e}[2j]] \\
&\leq \exp(\epsilon * |J|) \prod_{j=1}^{|\vec{e}|/2} \Pr[\kappa_{\vec{e}[2j-1]}(\mathsf{toMultiset}(\eta(\vec{i}_2, j))) = \vec{e}[2j]] \\
&= \exp(\epsilon * |J|) \Pr[\lfloor K(\vec{i})\rfloor_Q = \vec{e}] \\
&= \exp(t * \epsilon) \Pr[\lfloor K(\vec{i})\rfloor_Q \sqsupseteq \vec{e}]
\end{aligned}$$

since $|J| = t$. □

Much as Theorem 7 was related to sequential composition of differentially private functions, Theorem 8 is related to their "parallel composition" [17]. The parallel composition of differentially private functions is applying each of them to disjoint data sets. Unlike in the case of sequential composition where the privacy of the composite result is the sum of the privacy of each function, under parallel composition, the privacy of the composite result is the maximum ϵ over all the composed functions. Our above result, however, does not require disjointness of the data sets. Rather, we restrict each data point to being used by at most a fixed number t of functions and the composite result has $t * \epsilon$ privacy. Thus, our approach combines some aspects of sequential composition (multiple queries on each data point) with some of parallel composition (not all data points involved in all queries).

4 Unwinding for Automata

While general, dealing with trace distributions is difficult. Thus, we now fix a more structured model and show how to prove that such models have differential privacy. We chose to use a simplified version of probabilistic I/O automaton (cf. [14]).

4.1 Automata

Given a set S, let $\mathsf{Disc}(S)$ denote the set of discrete probability measures on S.

Definition 6. *A probabilistic automaton is a tuple $L = (S, s_0, I, O, T)$ where*

- *S is the set of states;*
- *s_0 is the unique initial state;*
- *I and O are countable and pairwise disjoint sets of actions, referred to as* input *and* output *actions respectively; and*
- *$T \subseteq S \times (I \cup O) \times \mathsf{Disc}(S)$ represent the possible transitions.*

As before, we use A for $I \cup O$ and partition the input set I into D, the set of data points, and Q, the set of queries, that is $I = D \cup Q$. When only one automaton is under consideration, we denote a transition $\langle s, a, \mu\rangle \in T$ by $s \xrightarrow{a} \mu$.

In the rest of this paper, we require that automata satisfy the following conditions:

- **Transition determinism:** For every state $s \in S$ and action $a \in A$, there is at most one $\mu \in \mathsf{Disc}(S)$ such that $s \xrightarrow{a} \mu$.

- **Output determinism:** For every state $s \in S$, output $o \in O$, action $a \in A$, and $\mu \in \mathsf{Disc}(S)$, if $s \xrightarrow{o} \mu$ and $s \xrightarrow{a} \mu'$, then $a = o$ and $\mu' = \mu$.
- **Quasi-input enabling:** For every state $s \in S$, inputs i_1 and i_2 in I, and $\mu_1 \in \mathsf{Disc}(S)$, if $s \xrightarrow{i_1} \mu_1$, then there exists μ_2 such that $s \xrightarrow{i_2} \mu_2$.

Output determinism and quasi-input enabling means that the state space may be partitioned into two parts: states that accept all of the inputs and states that produce exactly one output. Intuitively, we require that each output producing state produces only one output. In other words, we make the simplifying assumption that the choice of which output to perform is made by the automaton since an external non-determinism resolution mechanism might resolve such choices in a way that leaks information about the data set. These assumptions allow us to avoid modeling schedulers explicitly since fixing an input sequence resolves all the non-determinism. (e.g., see [14]). Owing to transition determinism, we will often write $s \xrightarrow{a} \mu$ without explicitly quantifying over μ (i.e., we treat $s \xrightarrow{a} \cdot$ as a partial function to μ).

We extend $\rightarrow$ to work over sequences of actions $\vec{a}$ as follows: $s \xrightarrow{[]} s'$ only if $s = s'$ and $s \xrightarrow{a:\vec{a}} s'$ iff $s \xrightarrow{a} \mu$ and there exists $s'' \in \mathsf{Supp}(\mu)$ such that $s'' \xrightarrow{\vec{a}} s'$. We say that s' is *reachable* from s iff there exists some $\vec{a}$ in A^* such that $s \xrightarrow{\vec{a}} s'$.

Given such an automaton L, we define $L_s(\vec{i})$ to be a distribution over traces (a sequence of actions) that gives the probability of seeing a prefix $\vec{a}$ of actions A given that automaton starts in s and the available inputs are $\vec{i}$.

$$
\begin{aligned}
L_s(i{:}\vec{i})(i{:}\vec{a}) &= \sum_{s' \in S} \mu(s') L_{s'}(\vec{i})(\vec{a}) && \text{if } s \xrightarrow{i} \mu \text{ and } i \in I \\
L_s(\vec{i})(o{:}\vec{a}) &= \sum_{s' \in S} \mu(s') L_{s'}(\vec{i})(\vec{a}) && \text{if } s \xrightarrow{o} \mu \text{ and } o \in O \\
L_s(\vec{i})([]) &= 1 \\
L_s(\vec{i})(\vec{a}) &= 0 && \text{otherwise}
\end{aligned}
$$

Lemma 1. *For all $\vec{a}$ in A^*, states s, and $\vec{i}$ in I^*, $L_s(\vec{i})(\vec{a})$ is well defined and between 0 and 1.*

Proof. Proof by induction over the structure of $\vec{a}$.

Case: $\vec{a} = []$. $L_s(\vec{i})(\vec{a})$ is 1.

Case: $\vec{a} = i{:}\vec{a}'$. If there does not exist $\vec{i}'$ such that $\vec{i} = i{:}\vec{i}'$ and $s \xrightarrow{i} \mu$, then $L_s(\vec{i})(\vec{a}) = 0$. If there does exist such a $\vec{i}'$, then $L_s(\vec{i})(\vec{a}) = \sum_{s' \in S} \mu(s') L_{s'}(\vec{i}')(\vec{a}')$. By the inductive hypothesis, $L_{s'}(\vec{i}')(\vec{a}')$ is well defined and between 0 and 1 for all s'. Since μ is a distribution over states and the events of being in a state are mutually exclusive, $\sum_{s' \in S} \mu(s') = 1$. Let $s_{\mathsf{max}} = \arg\max_{s' \in S} L_{s'}(\vec{i}')(\vec{a}')$. $L_s(\vec{i})(\vec{a}) = \sum_{s' \in S} \mu(s') L_{s'}(\vec{i}')(\vec{a}') \leq \sum_{s' \in S} \mu(s') L_{s_{\mathsf{max}}}(\vec{i}')(\vec{a}') = L_{s_{\mathsf{max}}}(\vec{i}')(\vec{a}') \leq 1$.

Case: $\vec{a} = o{:}\vec{a}'$. If there does not exist μ such that $s \xrightarrow{o} \mu$, then $L_s(\vec{i})(\vec{a}) = 0$. If there does, then $L_s(\vec{i})(\vec{a}) = \sum_{s' \in S} \mu(s') L_{s'}(\vec{i})(\vec{a}')$ and we can use the inductive hypothesis as above. □

We use $\lfloor L_s(\vec{i}) \rfloor_E(\vec{e})$ to denote probability of the adversary seeing $\vec{e} \in E^*$ as a prefix given that the available inputs are $\vec{i}$, where $E = Q \cup O$, the set of observable actions. To calculate $\lfloor L_s(\vec{i}) \rfloor_E(\vec{e})$, consider the set $\gamma(\vec{e})$ of action sequences $\vec{a}$ such that $\vec{a}$ is both in $\lfloor \vec{e} \rceil_E^{-1}$ and ends with the last element of $\vec{e}$. That is, $\gamma(\vec{e}) = \{\, \vec{a} \in A^* \mid \lfloor \vec{a} \rfloor_E = \vec{e} \wedge \mathsf{last}(\vec{a}) = \mathsf{last}(\vec{e}) \,\}$ with the special case that $\gamma([]) = \{[]\}$. To calculate $\lfloor L_s(\vec{i}) \rfloor_E(\vec{e})$, we need not consider all $\vec{a}$ in $\lfloor \vec{e} \rceil_E^{-1}$. Rather, we may focus only on those in $\gamma(\vec{e})$ since every $\vec{a}$ in $\lfloor \vec{e} \rceil_E^{-1}$ will have a prefix in $\gamma(\vec{e})$. Since it is impossible to see two different prefixes from $\gamma(\vec{e})$ during the same execution (no element of $\gamma(\vec{e})$ is the prefix of another), they are mutually exclusive. Thus, $\lfloor L_s(\vec{i}) \rfloor_E(\vec{e}) = \sum_{\vec{a} \in \gamma(\vec{e})} L_s(\vec{i})(\vec{a})$.

4.2 Unwinding Partitioning

Goguen and Meseguer introduced unwinding relations to simplify proving that a system has noninterference [11]. We define an unwinding relation for trace differential privacy. The main result of this section (Theorem 9) roughly states that the existence of such an unwinding relation for a given automaton implies that it satisfies trace differential privacy. This task is complicated for two reasons: we must deal with probabilities and we must keep track of the privacy error bound ϵ. While previous work has dealt with approximate probabilistic simulation relations in the context of cryptographic protocols [22], the process tracking an privacy error bound differs (see Section 5 for additional details).

The first step is to define a way of grouping states of an automaton into equivalence classes (or parts) via a family of partition relations, one for each possible value of the privacy error bound. Intuitively, if two states are in an equivalence class, they produce identical probability distributions over traces. Since the unwinding relation will be used to compare two executions of an automaton (with input sequences that differ on one data point), we consider two such families of partition relations ($\mathcal{U}_1^{\cdot}$ and $\mathcal{U}_2^{\cdot}$). The family of functions $\beta^{\cdot}$ maps equivalence classes of states from the first execution to *related* ones in the second execution that exhibit approximately the same behavior (in the sense of strong trace differential privacy). These considerations motivate Definition 7.

Definition 7. *[$\epsilon^\dagger$-View Partition Family] Let L be the probabilistic automaton $\langle S, s_0, I, O, T\rangle$. We call a family of triplets $\langle \mathcal{U}_1^{\cdot}, \mathcal{U}_2^{\cdot}, \beta^{\cdot}\rangle$ indexed by the set $[0, \epsilon^\dagger]$ an* $\epsilon^\dagger$-view partition family *for L if for all ϵ in $[0, \epsilon^\dagger]$, $\mathcal{U}_1^\epsilon$ is a partition of some subset of the state space S, $\mathcal{U}_2^\epsilon$ is a partition of some subset of the state space S, and β^ϵ is a bijection from $\mathcal{U}_1^\epsilon$ to $\mathcal{U}_2^\epsilon$.*

Definition 8 roughly ensures that probability distributions over traces arising from two states, s_1 and s_2, in related equivalence classes are close (in the sense of strong trace differential privacy with error bound ϵ) by requiring that matching transitions in the two automata from those states have either (a) identical probability distributions and thus maintain the bound (the "maintenance" condition); or (b) they diverge by an error bound δ, in which case the requirements apply recursively to the resulting states with a smaller error bound $\epsilon' \leq \epsilon - \delta$ (the "degradation" condition). The form of δ involves natural logarithms because the privacy error bound in the differential privacy definition appears in the exponent. Since ϵ' can be any number in the set $[0, \epsilon]$, we require Definition 7 to refer to an indexed family of partition relations equipped with a bijective function.

Since the events of a being in a state are mutually disjoint, we raise a distribution μ over states to sets of states as follows: $\mu(S') = \sum_{s \in S'} \mu(s)$.

Definition 8. *[$\epsilon^\dagger$-Unwinding Relation] Let L be the probabilistic automaton $\langle S, s_0, I, O, T\rangle$. For non-negative real numbers $\epsilon^\dagger$, a $\epsilon^\dagger$-view partition family $\langle \mathcal{U}_1^{\cdot}, \mathcal{U}_2^{\cdot}, \beta^{\cdot}\rangle$ is an* $\epsilon^\dagger$-unwinding relation *for L if for all ϵ in $[0, \epsilon^\dagger]$, parts U_1 of $\mathcal{U}_1^\epsilon$, s_1 in U_1, s_2 in $\beta^\epsilon(U_1)$, a in A, and μ_1 in $\mathsf{Disc}(S)$, $s_1 \xrightarrow{a} \mu_1$ iff $s_2 \xrightarrow{a} \mu_2$ for some μ_2 in $\mathsf{Disc}(S)$ and whenever $s_1 \xrightarrow{a} \mu_1$, there exists μ_2 in $\mathsf{Disc}(S)$ such that $s_2 \xrightarrow{a} \mu_2$ either*

- *Maintenance: $\mathcal{U}_1^\epsilon$ partitions a superset of $\mathsf{Supp}(\mu_1)$, $\mathcal{U}_2^\epsilon$ partitions a superset of $\mathsf{Supp}(\mu_2)$, and for all parts U_1' of $\mathcal{U}_1^\epsilon$, $\mu_1(U_1') = \mu_2(\beta^\epsilon(U_1'))$; or*
- *Degradation: there exists a non-negative real number ϵ' in $[0, \epsilon]$ such that $\mathcal{U}_1^{\epsilon'}$ partitions a superset of $\mathsf{Supp}(\mu_1)$, $\mathcal{U}_2^{\epsilon'}$ partitions a superset of $\mathsf{Supp}(\mu_2)$, and for all parts U_1' of $\mathcal{U}_1^{\epsilon'}$, $\epsilon' \leq \epsilon - \ln(\mu_1(U_1')) + \ln(\mu_2(\beta^{\epsilon'}(U_1')))$ and $\epsilon' \leq \epsilon + \ln(\mu_1(U_1')) - \ln(\mu_2(\beta^{\epsilon'}(U_1')))$.*

Lemma 2. *For all $\epsilon^\dagger$-unwinding relations $\langle \mathcal{U}_1^{\cdot}, \mathcal{U}_2^{\cdot}, \beta^{\cdot}\rangle$, ϵ in $[0, \epsilon^\dagger]$, parts U_1 in $\mathcal{U}_1^\epsilon$, s_1 in U_1, s_2 in $\beta(U_1)$, $\vec{i}$ in I^*, and $\vec{a}$ in A^*, $L_{s_1}(\vec{i})(\vec{a}) \leq \exp(\epsilon) L_{s_2}(\vec{i})(\vec{a})$ and $L_{s_2}(\vec{i})(\vec{a}) \leq \exp(\epsilon) L_{s_1}(\vec{i})(\vec{a})$.*

Proof. Proof by induction over the structure of $\vec{a}$.

Case: $\vec{a} = []$. $L_{s_1}(\vec{i})(\vec{a}) = 1 = L_{s_2}(\vec{i})(\vec{a})$.

Case: $\vec{a} = i{:}\vec{a}'$ where $i \in I$. We consider the following subcases:

- Subcase: $\vec{i} = []$. $L_{s_1}([])(i{:}\vec{a}') = 0 = L_{s_2}([])(i{:}\vec{a}')$.

- Subcase: $\vec{i} = i{:}\vec{i'}$ for some $\vec{i'}$ and $s_1 \xrightarrow{i} \mu_1$. $L_{s_1}(i{:}\vec{i'})(i{:}\vec{a'}) = \sum_{s'_1 \in S} \mu_1(s'_1) L(s'_1, \vec{i'})(\vec{a'})$. Furthermore, since $\langle \mathcal{U}_1, \mathcal{U}_2, \beta^{\cdot} \rangle$ is an $\epsilon^\dagger$-unwinding relation and ϵ is in $[0, \epsilon^\dagger]$, there exists μ_2 such that $s_2 \xrightarrow{i} \mu_2$. Since $s_2 \xrightarrow{i} \mu_2$, $L_{s_2}(i{:}\vec{i'})(i{:}\vec{a'}) = \sum_{s'_2 \in S} \mu_2(s'_2) L(s'_2, \vec{i'})(\vec{a'})$. Now we consider two subsubcases:

  Subsubcase: Maintenance. In this case, $\mathcal{U}_1^\epsilon$ partitions a superset of $\mathsf{Supp}(\mu_1)$, $\mathcal{U}_2^\epsilon$ partitions a superset of $\mathsf{Supp}(\mu_2)$, and for all parts U'_1 of $\mathcal{U}_1^\epsilon$, $\mu_1(U'_1) = \mu_2(\beta^\epsilon(U'_1))$. Thus,

$$
\begin{aligned}
L_{s_1}(i{:}\vec{i'})(i{:}\vec{a'}) &= \sum_{s'_1 \in S} \mu_1(s'_1) L_{s'_1}(\vec{i'})(\vec{a'}) && (1)\\
&= \sum_{U_1 \in \mathcal{U}_1^\epsilon} \sum_{s'_1 \in U_1} \mu_1(s'_1) L_{s'_1}(\vec{i'})(\vec{a'}) && (2)\\
&\leq \sum_{U_1 \in \mathcal{U}_1^\epsilon} \sum_{s'_1 \in U_1} \mu_1(s'_1) L_{s_{\mathsf{max}}(U_1)}(\vec{i'})(\vec{a'}) && (3)\\
&= \sum_{U_1 \in \mathcal{U}_1^\epsilon} \mu_1(U_1) L_{s_{\mathsf{max}}(U_1)}(\vec{i'})(\vec{a'}) && (4)\\
&\leq \sum_{U_1 \in \mathcal{U}_1^\epsilon} \mu_1(U_1) \exp(\epsilon) L_{s_{\mathsf{min}}(\beta^\epsilon(U_1))}(\vec{i'})(\vec{a'}) && (5)\\
&= \exp(\epsilon) \sum_{U_1 \in \mathcal{U}_1^\epsilon} \mu_2(\beta^\epsilon(U_1)) L_{s_{\mathsf{min}}(\beta^\epsilon(U_1))}(\vec{i'})(\vec{a'}) && (6)\\
&= \exp(\epsilon) \sum_{U_1 \in \mathcal{U}_1^\epsilon} \sum_{s'_2 \in \beta^\epsilon(U_1)} \mu_2(s'_2) L_{s_{\mathsf{min}}(\beta^\epsilon(U_1))}(\vec{i'})(\vec{a'}) && (7)\\
&\leq \exp(\epsilon) \sum_{U_1 \in \mathcal{U}_1^\epsilon} \sum_{s'_2 \in \beta^\epsilon(U_1)} \mu_2(s'_2) L_{s'_2}(\vec{i'})(\vec{a'}) && (8)\\
&= \exp(\epsilon) \sum_{U_2 \in \mathcal{U}_2^\epsilon} \sum_{s'_2 \in U_2} \mu_2(s'_2) L_{s'_2}(\vec{i'})(\vec{a'}) && (9)\\
&= \exp(\epsilon) \sum_{s' \in S} \mu_2(s') L_{s'}(\vec{i'})(\vec{a'}) && (10)\\
&= \exp(\epsilon) L_{s_2}(i{:}\vec{i'})(i{:}\vec{a'}) && (11)
\end{aligned}
$$

  where $s_{\mathsf{max}}(U_1) = \mathsf{arg}\max_{s'_1 \in U_1} L_{s'}(\vec{i'})(\vec{a'})$ and $s_{\mathsf{min}}(\beta^\epsilon(U_1)) = \mathsf{arg}\min_{s'_1 \in \beta^\epsilon(U_1)} L_{s'_1}(\vec{i'})(\vec{a'})$. Line 2 follows from the fact that $\mathcal{U}_1^\epsilon$ partitions a superset of $\mathsf{Supp}(\mu_1)$ and Line 10 follows from the fact that $\mathcal{U}_2^{\epsilon'}$ partitions a superset of $\mathsf{Supp}(\mu_2)$. Line 5 follows from the the inductive hypothesis. Line 9 follows from $\beta^{\epsilon'}$ being a bijection. Proving that $L_{s_2}(\vec{i})(\vec{a}) \leq \exp(\epsilon) L_{s_1}(\vec{i})(\vec{a})$ is almost the same reversing the roles of s_1 and s_2.

  Subsubcase: Degradation. In this case, there exists ϵ' and μ_2 such that $s_2 \xrightarrow{a} \mu_2$ and for all parts U'_1 in $\mathcal{U}_1^{\epsilon'}$, $\epsilon' \leq \epsilon - (\ln(\mu_1(U'_1)) - \ln(\mu_2(\beta(U'_1))))$. By the inductive hypothesis, for all U'_1 in $\mathcal{U}_1^{\epsilon'}$, s'_1 in U'_1,

and s_2' in $\beta(U_1')$, $L_{s_1'}(\vec{i'})(\vec{a'}) \leq \exp(\epsilon')L_{s_2'}(\vec{i'})(\vec{a'})$. Thus,

$$
\begin{aligned}
&L_{s_1}(i{:}\vec{i'})(i{:}\vec{a'}) && (12)\\
&\quad= \sum_{s_1' \in S} \mu_1(s_1')L_{s_1'}(\vec{i'})(\vec{a'}) && (13)\\
&\quad= \sum_{U_1' \in \mathcal{U}_1^{\epsilon'}} \sum_{s_1' \in U_1'} \mu_1(s_1')L_{s_1'}(\vec{i'})(\vec{a'}) && (14)\\
&\quad\leq \sum_{U_1' \in \mathcal{U}_1^{\epsilon'}} \sum_{s_1' \in U_1'} \mu_1(s_1')L_{s_{\mathsf{max}}(U_1')}(\vec{i'})(\vec{a'}) && (15)\\
&\quad= \sum_{U_1' \in \mathcal{U}_1^{\epsilon'}} \mu_1(U_1')L_{s_{\mathsf{max}}(U_1')}(\vec{i'})(\vec{a'}) && (16)\\
&\quad\leq \sum_{U_1' \in \mathcal{U}_1^{\epsilon'}} \mu_1(U_1')\exp(\epsilon')L_{s_{\mathsf{min}}(\beta^{\epsilon'}(U_1'))}(\vec{i'})(\vec{a'}) && (17)\\
&\quad= \sum_{U_1' \in \mathcal{U}_1^{\epsilon'}} \frac{\mu_1(U_1')}{\mu_2(\beta^{\epsilon'}(U_1'))}\mu_2(\beta^{\epsilon'}(U_1'))\exp(\epsilon')L_{s_{\mathsf{min}}(\beta^{\epsilon'}(U_1'))}(\vec{i'})(\vec{a'}) && (18)\\
&\quad= \sum_{U_1' \in \mathcal{U}_1^{\epsilon'}} \exp(\ln(\mu_1(U_1')) - \ln(\mu_2(\beta^{\epsilon'}(U_1'))))\mu_2(\beta^{\epsilon'}(U_1'))\exp(\epsilon')L_{s_{\mathsf{min}}(\beta^{\epsilon'}(U_1'))}(\vec{i'})(\vec{a'}) && (19)\\
&\quad= \sum_{U_1' \in \mathcal{U}_1^{\epsilon'}} \exp(\ln(\mu_1(U_1')) - \ln(\mu_2(\beta^{\epsilon'}(U_1'))))\exp(\epsilon') \sum_{s_2' \in \beta^{\epsilon'}(U_1')} \mu_2(s_2')L_{s_{\mathsf{min}}(\beta^{\epsilon'}(U_1'))}(\vec{i'})(\vec{a'}) && (20)\\
&\quad\leq \sum_{U_1' \in \mathcal{U}_1^{\epsilon'}} \exp(\epsilon' + \ln(\mu_1(U_1')) - \ln(\mu_2(\beta^{\epsilon'}(U_1')))) \sum_{s_2' \in \beta^{\epsilon'}(U_1')} \mu_2(s_2')L_{s_2'}(\vec{i'})(\vec{a'}) && (21)\\
&\quad\leq \sum_{U_1' \in \mathcal{U}_1^{\epsilon'}} \exp(\epsilon) \sum_{s_2' \in \beta^{\epsilon'}(U_1')} \mu_2(s_2')L_{s_2'}(\vec{i'})(\vec{a'}) && (22)\\
&\quad= \exp(\epsilon) \sum_{U_2' \in \mathcal{U}_2^{\epsilon'}} \sum_{s_2' \in U_2'} \mu_2(s_2')L_{s_2'}(\vec{i'})(\vec{a'}) && (23)\\
&\quad= \exp(\epsilon) \sum_{s' \in S} \mu_2(s')L_{s'}(\vec{i'})(\vec{a'}) && (24)\\
&\quad= \exp(\epsilon)L_{s_2}(i{:}\vec{i'})(i{:}\vec{a'}) && (25)
\end{aligned}
$$

where $s_{\mathsf{max}}(U_1) = \mathsf{arg}\max_{s_1' \in U_1} L_{s'}(\vec{i'})(\vec{a'})$ and $s_{\mathsf{min}}(\beta^{\epsilon}(U_1)) = \mathsf{arg}\min_{s_1' \in \beta^{\epsilon}(U_1)} L_{s_1'}(\vec{i'})(\vec{a'})$. Line 14 follows from the fact that $\mathcal{U}_1^{\epsilon'}$ partitions a superset of $\mathsf{Supp}(\mu_1)$ and Line 24 follows from the fact that $\mathcal{U}_2^{\epsilon'}$ partitions a superset of $\mathsf{Supp}(\mu_2)$. Line 17 follows from the the inductive hypothesis. Line 19 follows from the fact that for every $U' \in \mathcal{U}'$, $\mu_1(U')/\mu_2(U') = \exp(\ln(\mu_1(U')))/\exp(\ln(\mu_2(U')))$. Line 22 follows from the fact that for every $U' \in \mathcal{U}'$, $\epsilon' \leq \epsilon - (\ln(\mu_1(U')) - \ln(\mu_2(U')))$. Line 23 follows from $\beta^{\epsilon'}$ being a bijection. Proving that $L_{s_2}(\vec{i})(\vec{a}) \leq \exp(\epsilon)L_{s_1}(\vec{i})(\vec{a})$ is almost the same reversing the roles of s_1 and s_2.

- Subcase: $\vec{i} = i{:}\vec{i'}$ for some $\vec{i'}$ but there exists no μ_1 such that $s_1 \xrightarrow{i} \mu_1$. There also cannot exist a μ_2 such that $s_2 \xrightarrow{i} \mu_2$. Thus, $L_{s_1}(\vec{i})(i{:}\vec{a'}) = 0 = L_{s_2}(\vec{i})(i{:}\vec{a'})$.

Case: $\vec{a} = o{:}\vec{a'}$ where $o \in O$. Mostly as above. □

Theorem 9. *If for every state s' reachable from s, and $d \in D$, $s' \xrightarrow{d} \mu$ implies that there exists an ϵ-unwinding relation $\langle \mathcal{U}_1, \mathcal{U}_2, \beta^{\cdot} \rangle$ such that $\mu(\beta^{\epsilon}(U_{s'})) = 1$ where $U_{s'}$ is the part in $\mathcal{U}_1^{\epsilon}$ that has s' in it (i.e., $s' \in U_{s'} \in \mathcal{U}_1^{\epsilon}$), then L_s has ϵ-strong trace differential privacy.*

Proof. Arbitrarily fix $\vec{i}_1$ and $\vec{i}_2$ such that $\Delta(\vec{i}_1, \vec{i}_2) = 1$.

Now, we must show that for all s and $\vec{e}$,

$$\lfloor L_s(\vec{i}_1) \rfloor_E(\vec{e}) = \sum_{\vec{a} \in \gamma(\vec{e})} L_s(\vec{i}_1)(\vec{a}) \leq \exp(\epsilon) \sum_{\vec{a} \in \gamma(\vec{e})} L_s(\vec{i}_2)(\vec{a}) = \lfloor L_s(\vec{i}_2) \rfloor_E(\vec{e})$$

We use induction over the structures of $\vec{i}_1$, $\vec{i}_2$, and $\vec{e}$.

Case: $\vec{e} = []$. Only in the case where $\vec{e} = []$, can $[]$ be in $\gamma(\vec{e})$. In this case, $\lfloor L_s(\vec{i}_1) \rfloor_E([]) = 1 = \lfloor L_s(\vec{i}_2) \rfloor_E([])$ irrespective of $\vec{i}_1$ and $\vec{i}_2$. (We assume that $\vec{e} \neq []$ in the reminder of this proof.)

Case: $\vec{i}_1 = []$ and $\vec{i}_2 = []$. $L_s([])(\vec{a}) = L_s([])(\vec{a})$ for all $\vec{a} \in \gamma(\vec{e})$ for any $\vec{e}$.

Case: $\vec{i}_1 = d{:}\vec{i}'_1$ and $\vec{i}_2 = d{:}\vec{i}'_2$. We consider three mutually exclusive subcases.

- Subcase: $s \xrightarrow{d} \mu$. For all $\vec{a}$ such that for no $\vec{a}'$, $\vec{a} = d{:}\vec{a}'$, $L_s(\vec{i}_1)(\vec{a}) = 0 = L_s(\vec{i}_2)(\vec{a})$. Since such $\vec{a}$ add nothing to the summations, we may ignore them and limit our attention to $\vec{a} = d{:}\vec{a}'$ in $\gamma(\vec{e})$. Note that all such $\vec{a}'$ are in $\gamma(\vec{e})$ iff $d{:}\vec{a}'$ is in $\gamma(\vec{e})$.

  For all states $s' \in \mathsf{Supp}(\mu)$, every state s'' reachable from s' will have a unwinding relation ϵ-unwinding relation $\langle \mathcal{U}_1, \mathcal{U}_2, \beta^{\cdot} \rangle$ such that $\mu(\beta^{\epsilon}(U_{s''})) = 1$ where $U_{s''}$ is the part in $\mathcal{U}_1^{\epsilon}$ that holds s'' since s'' is reachable from s. Thus, we may apply the inductive hypothesis to $\vec{a}'$ to get that for all $s' \in \mathsf{Supp}(\mu)$, $\sum_{\vec{a}' \in \gamma(\vec{e})} L_{s'}(\vec{i}'_1)(\vec{a}') \leq \exp(\epsilon) \sum_{\vec{a}' \in \gamma(\vec{e})} L_{s'}(\vec{i}'_2)(\vec{a}')$. Considering the sum over all such $\vec{a}$, we get

$$\begin{aligned}
\sum_{\vec{a} \in \gamma(\vec{e})} L_s(\vec{i}_1)(\vec{a}) &= \sum_{d:\vec{a}' \in \gamma(\vec{e})} L_s(\vec{i}_1)(d{:}\vec{a}') \\
&= \sum_{\vec{a}' \in \gamma(\vec{e})} \sum_{s' \in S} \mu(s') L_{s'}(\vec{i}'_1)(\vec{a}') \\
&= \sum_{s' \in S} \mu(s') \sum_{\vec{a}' \in \gamma(\vec{e})} L_{s'}(\vec{i}'_1)(\vec{a}') \\
&= \sum_{s' \in \mathsf{Supp}(\mu)} \mu(s') \sum_{\vec{a}' \in \gamma(\vec{e})} L_{s'}(\vec{i}'_1)(\vec{a}') \\
&\leq \sum_{s' \in \mathsf{Supp}(\mu)} \mu(s') \exp(\epsilon) \sum_{\vec{a}' \in \gamma(\vec{e})} L_{s'}(\vec{i}'_2)(\vec{a}') \\
&= \exp(\epsilon) \sum_{\vec{a}' \in \gamma(\vec{e})} \sum_{s' \in S} \mu(s') L_{s'}(\vec{i}'_2)(\vec{a}') \\
&= \exp(\epsilon) \sum_{d:\vec{a}' \in \gamma(\vec{e})} L_s(\vec{i}_2)(d{:}\vec{a}') \\
&= \exp(\epsilon) \sum_{\vec{a} \in \gamma(\vec{e})} L_s(\vec{i}_2)(\vec{a})
\end{aligned}$$

  where $d{:}\vec{a}'$ in the expression $d{:}\vec{a}' \in \gamma(\vec{e})$ ranges over only those elements of $\gamma(\vec{e})$ of the form $d{:}\vec{a}'$. That is, $\sum_{d:\vec{a}' \in \gamma(\vec{e})}$ is shorthand for $\sum_{d:\vec{a}' \in \{\, \vec{a}'' \in \gamma(\vec{e}) \;\mid\; \exists \vec{a}' \in A^* \text{ s.t. } d:\vec{a}' = \vec{a}'' \,\}}$. Note that the last line follows from the fact that $L_s(\vec{i}_2)(\vec{a}) = 0$ for all $\vec{a}$ not of the form $d{:}\vec{a}'$.

- Subcase: $s \xrightarrow{o} \mu$. For all $\vec{a}$ such that for no $\vec{a}'$ such that $\vec{a} = o{:}\vec{a}'$, $L_s(\vec{i}_1)(\vec{a}) = 0 = L_s(\vec{i}_2)(\vec{a})$. Since such $\vec{a}$ add nothing to the summations, we may ignore them and limit our attention to $o{:}\vec{a}'$ in $\gamma(\vec{e})$. Unless $\vec{e} = o{:}\vec{e}'$ for some $\vec{e}'$, no such $o{:}\vec{a}'$ will be in $\gamma(\vec{e})$ and both summations will be zero. Thus, we limit our attention to the case where $\vec{e} = o{:}\vec{e}'$ for some $\vec{e}'$. In this case, we may use the inductive hypothesis on

$\vec{e}'$ to get that for all $s' \in \mathsf{Supp}(\mu)$, $\sum_{\vec{a}' \in \gamma(\vec{e}')} L_{s'}(\vec{i}_1)(\vec{a}') \leq \exp(\epsilon) \sum_{\vec{a}' \in \gamma(\vec{e}')} L_{s'}(\vec{i}_2)(\vec{a}')$. Thus,

$$\begin{aligned}
\sum_{\vec{a} \in \gamma(\vec{e})} L_s(\vec{i}_1)(\vec{a}) &= \sum_{o:\vec{a}' \in \gamma(o:\vec{e}')} L_s(\vec{i}_1)(o{:}\vec{a}') \\
&= \sum_{\vec{a}' \in \gamma(\vec{e}')} \sum_{s' \in S} \mu(s') L_{s'}(\vec{i}_1)(\vec{a}') \\
&= \sum_{s' \in \mathsf{Supp}(\mu)} \mu(s') \sum_{\vec{a}' \in \gamma(\vec{e}')} L_{s'}(\vec{i}_1)(\vec{a}') \\
&\leq \sum_{s' \in \mathsf{Supp}(\mu)} \mu(s') \exp(\epsilon) \sum_{\vec{a}' \in \gamma(\vec{e}')} L_{s'}(\vec{i}_2)(\vec{a}') \\
&= \exp(\epsilon) \sum_{\vec{a}' \in \gamma(\vec{e}')} \sum_{s' \in S} \mu(s') L_{s'}(\vec{i}_2)(\vec{a}') \\
&= \exp(\epsilon) \sum_{o:\vec{a}' \in \gamma(o:\vec{e}')} L_s(\vec{i}_2)(o{:}\vec{a}') \\
&= \exp(\epsilon) \sum_{\vec{a} \in \gamma(\vec{e})} L_s(\vec{i}_2)(\vec{a})
\end{aligned}$$

- Subcase: Otherwise. Since $\vec{a} \neq []$, $L_s(\vec{i}_1)(\vec{a}) = 0 = L_s(\vec{i}_2)(\vec{a})$ for all $\vec{a} \in \gamma(\vec{e})$.

Case: $\vec{i}_1 = q{:}\vec{i}'_1$ and $\vec{i}_2 = q{:}\vec{i}'_2$. Much as above just using that $\vec{a}$ is only in $\gamma(\vec{e})$ if $\vec{e} = i{:}\vec{e}'$ for some $\vec{e}'$ and $\vec{a}' \in \gamma(\vec{e}')$.

Case: $\vec{i}_2 = d{:}\vec{i}_1$. We consider the following subcases.

- Subcase: $s \xrightarrow{d} \mu$. Since s is reachable from s, there exists an ϵ-unwinding relation $\langle \mathcal{U}_1, \mathcal{U}_2, \beta^{\cdot} \rangle$ such that $\mu(\beta^{\epsilon}(U_s)) = 1$ where $s \in U_s$. For every $\vec{a}$ in $\gamma(\vec{e})$, we show that $d{:}\vec{a}$ is such that $L_s(\vec{i}_1)(\vec{a}) \leq \exp(\epsilon) L_s(\vec{i}_2)(d{:}\vec{a})$.

$$\begin{aligned}
L_s(\vec{i}_1)(\vec{a}) &\leq \exp(\epsilon) L_{s_{\mathsf{min}}}(\vec{i}_1)(\vec{a}) && (26) \\
&= \exp(\epsilon) \left(\sum_{s' \in \beta^{\epsilon}(U_s)} \mu(s') \right) L_{s_{\mathsf{min}}}(\vec{i}_1)(\vec{a}) && (27) \\
&\leq \exp(\epsilon) \sum_{s' \in \beta^{\epsilon}(U_s)} \mu(s') L_{s'}(\vec{i}_1)(\vec{a}) && (28) \\
&= \exp(\epsilon) \sum_{s' \in S} \mu(s') L_{s'}(\vec{i}_1)(\vec{a}) && (29) \\
&= \exp(\epsilon) L_s(d{:}\vec{i}_1)(d{:}\vec{a}) && (30) \\
&= \exp(\epsilon) L_s(\vec{i}_2)(d{:}\vec{a}) && (31)
\end{aligned}$$

  where the first line follows from Lemma 2 with $s_{\mathsf{min}} = \mathsf{arg}\min_{s' \in \beta^{\epsilon}(U_s)} L_{s'}(\vec{i}_1)(\vec{a})$. For the second line, recall that $\mu(\beta^{\epsilon}(U_s)) = \sum_{s' \in \beta^{\epsilon}(U_s)} \mu(s') = 1$. For Line 29, recall that $\mu(s') = 0$ for all $s' \notin \beta^{\epsilon}(U_s)$. Thus, since for every $\vec{a}$ in $\gamma(\vec{e})$, $d{:}\vec{a}$ is also in $\gamma(\vec{e})$,

$$\sum_{\vec{a} \in \gamma(\vec{e})} L_s(\vec{i}_1)(\vec{a}) \leq \sum_{d:\vec{a} \in \gamma(\vec{e})} \exp(\epsilon) L_s(\vec{i}_2)(d{:}\vec{a}) \leq \exp(\epsilon) \sum_{\vec{a} \in \gamma(\vec{e})} L_s(\vec{i}_2)(\vec{a})$$

- Subcase: $s \xrightarrow{o} \mu$. As in the corresponding subcase in the case for $\vec{i}_1 = d{:}\vec{i}'_1$ and $\vec{i}_2 = d{:}\vec{i}'_2$, we may ignore $\vec{a}$ not of the form $\vec{a} = o{:}\vec{a}'$ and $\vec{e}$ not of the form $o{:}\vec{e}'$. In this case, we may use the inductive hypothesis on $\vec{e}'$ as before to get the required result.

- Subcase: Otherwise. Since s does not transition under d in this case, and the automaton has quasi-input enabling, it does not transition under any action. Thus, since $\vec{a} \neq []$, $L_s(\vec{i_1})(\vec{a}) = 0$ for all $\vec{a} \in \gamma(\vec{e})$.

Case: $\vec{i_1} = d{:}\vec{i_2}$. We consider the following subcases.

- Subcase: $s \xrightarrow{d} \mu$. $L_s(\vec{i_1})(\vec{a}) = L_s(d{:}\vec{i_2})(\vec{a}) = 0$ unless $\vec{a} = d{:}\vec{a}'$ for some $\vec{a}'$ since s cannot transition under outputs and the next input is d. Thus,

$$\begin{aligned}
\lfloor L_s(\vec{i_1})\rfloor_E(\vec{e}) &= \lfloor L_s(d{:}\vec{i_2})\rfloor_E(\vec{e}) && (32)\\
&= \sum_{\vec{a}\in\gamma(\vec{e})} L_s(d{:}\vec{i_2})(\vec{a}) && (33)\\
&= \sum_{d:\vec{a}'\in\gamma(\vec{e})} L_s(d{:}\vec{i_2})(d{:}\vec{a}') && (34)\\
&= \sum_{d:\vec{a}'\in\gamma(\vec{e})} \sum_{s'\in S} \mu(s') L_{s'}(\vec{i_2})(\vec{a}') && (35)\\
&= \sum_{d:\vec{a}'\in\gamma(\vec{e})} \sum_{s'\in\beta^\epsilon(U_s)} \mu(s') L_{s'}(\vec{i_2})(\vec{a}') && (36)\\
&\leq \sum_{d:\vec{a}'\in\gamma(\vec{e})} \sum_{s'\in\beta^\epsilon(U_s)} \mu(s') \exp(\epsilon) L_{s_{\mathsf{min}}}(\vec{i_2})(\vec{a}') && (37)\\
&= \exp(\epsilon) \sum_{d:\vec{a}'\in\gamma(\vec{e})} L_{s_{\mathsf{min}}}(\vec{i_2})(\vec{a}') && (38)\\
&\leq \exp(\epsilon) \sum_{d:\vec{a}'\in\gamma(\vec{e})} L_s(\vec{i_2})(\vec{a}') && (39)\\
&\leq \exp(\epsilon) \sum_{\vec{a}\in\gamma(\vec{e})} L_s(\vec{i_2})(\vec{a}) && (40)\\
&= \exp(\epsilon) \lfloor L_s(\vec{i_2})\rfloor_E(\vec{e}) && (41)
\end{aligned}$$

  where $s_{\mathsf{min}} = \mathsf{arg}\min_{s'\in\beta^\epsilon(U_s)} L_{s'}(\vec{i_1})(\vec{a}')$. Line 36 follows from $\mu(s') = 0$ for all s' not in $\beta^\epsilon(U_s)$. Line 37 follows from Lemma 2. Line 38 follows from $\mu(\beta^\epsilon(U_s)) = 1$.

- Subcase: $s \xrightarrow{o} \mu$. As above in the other subcases for $s \xrightarrow{o} \mu$.

- Subcase: Otherwise. In the case where $s \xrightarrow{d} \mu$ for no μ and $\vec{a} \neq []$, everything is 0, which must be lower than any possible value of $\exp(\epsilon)\lfloor L_s(\vec{i_2})\rfloor_E(\vec{e})$.

□

4.3 Demonstrating Unwinding

We describe a system that is a synchronous database implementation of $\varkappa$ individually capped at $t * \epsilon^\dagger$. This system has many similarities with PINQ [17]. Like PINQ, it answers queries in the same order as it receives them. It does not model some aspects of PINQ, e.g., PINQ offers transformations that converts one data set into a new related data set by way of SQL-like transformations (e.g., `where` and `select`). However, unlike PINQ, our system has a mutable database and can provide tight privacy error bounds by removing overused data points.

We prove that this system has $(t * \epsilon^\dagger)$-strong trace differential privacy. One way to do this would be to prove that it is indeed a synchronous database implementation of $\varkappa$ individually capped at $t * \epsilon^\dagger$ and then use Theorem 8. However, it is unclear how we could prove this property. Thus, we instead show an unwinding relation for it and use Theorem 9.

System. The source code for such a system could look as follows:

```
datapoints := new Array(t);
for(int j := 0; j < t; j++){
   datapoints[j] := new LinkedList();
}
curSlot := 0;
while(1){
   x := input();
   if(datapoint(x)){
      datapoints[curSlot].add(x);
   }else{
      k := get_santitzation_function(x);
      print(k.compute(to_multiset(datapoints)));
      curSlot := (curSlot + 1) mod t;
      delete datapoints[curSlot];
      datapoints[curSlot] := new LinkedList();
   }
}
```

Intuitively, this program keeps t lists of data points. If an input is a data point, it is added to the list stored at the current value of `curSlot`. If it is a query, all the data points in these lists are put into one large multiset and the query requested by the input is computed on that multiset. Furthermore, the index `curSlot` to one of these lists is cyclically shifted and the linked list to which it now points is replaced with the empty linked list. Since there are only t slots, this means that each linked list will only last for t queries before being deleted. Since each query has $\epsilon^\dagger$-differential privacy, this ensures that each data point will only be involved in $t * \epsilon^\dagger$ worth of queries.

Note that if $t = 0$, we take the program to have an array `datapoints` of length 0, in which case it never stores any data points and `to_multiset(datapoints)` is always the empty set.

Automaton Model. To formally prove that this program has $(t * \epsilon^\dagger)$-strong trace differential privacy, we must give an automaton model of the program. We model the array `datapoints` as a t-tuple of multisets of data points. We model the index `curSlot` as an integer c ranging from 1 to t, which selects one of the multisets of the t-tuple. The state must also keep track of a program counter pc, which has the value in if the program is waiting for input and the value out if the program is about to produce output. Lastly, the state keeps track of which output from O it is about to produce when the pc is out. Thus, the set of states is $(\mathcal{M}(D))^t \times \{1, \ldots, t\} \times \{\mathsf{in}, \mathsf{out}\} \times O$ where $\mathcal{M}(D)$ is the set of all multisets with elements from D.

The initial state s_0 is $\langle\langle \{\!\{\}\!\}_1, \ldots, \{\!\{\}\!\}_t\rangle, 1, \mathsf{in}, o_0\rangle$ where we use $\langle \{\!\{\}\!\}_1, \ldots \{\!\{\}\!\}_t\rangle$ to represent a t-tuple of empty multisets and o_0 is an output arbitrarily selected to be in the initial state. (The value of o_0 does not matter since pc is in.)

To define the transition relation $\rightarrow$, we use the following definitions. Given a state $s = \langle\langle M_1, \ldots, M_t\rangle, c, pc, o\rangle$ and $d \in D$, let

$$\mathsf{add}(s, d) = \langle\langle M'_1, \ldots, M'_t\rangle, c, pc, o\rangle$$

where $M'_c = M_c \uplus \{d\}$ (using $\uplus$ for multiset union) and $M'_{c'} = M_{c'}$ for all $c' \neq c$. Where s is as before, for $o' \in O$, let

$$\mathsf{output}(s, o') = \langle\langle M'_1, \ldots, M'_t\rangle, (c+1) \bmod t, \mathsf{out}, o'\rangle$$

where $M'_{(c+1) \bmod t} = \{\!\{\}\!\}$ and $M'_{c'} = M_{c'}$ for all $c' \neq (c+1) \bmod t$ Let

$$\mathsf{wait}(s) = \langle\langle M_1, \ldots, M_t\rangle, c, \mathsf{in}, o\rangle$$

We also use *Dirac* distributions: let $\mathsf{Dirac}(s)$ be the distribution such that $\mathsf{Dirac}(s)(s) = 1$ and $\mathsf{Dirac}(s)(s') = 0$ for all $s' \neq s$. Given a query q in Q, we let κ_q be the sanitization function that answers that query.

The only transitions are those such that

- $s \xrightarrow{d} \mathsf{Dirac}(\mathsf{add}(s, c, d))$ for all d in D where the state s has the form $\langle\langle M_1, \ldots, M_t\rangle, c, \mathsf{in}, o\rangle$;
- $s \xrightarrow{q} \mu$ for all q in Q where the state s has the form $\langle\langle M_1, \ldots, M_t\rangle, c, \mathsf{in}, o\rangle$, and the distribution μ is such that $\mu(\mathsf{output}(s, o')) = \Pr[\kappa_q(\biguplus_{c'=1}^{t} M_{c'}) = o']$ and $\mu(s') = 0$ for all other states s'; and
- $s \xrightarrow{o} \mathsf{Dirac}(\mathsf{wait}(s))$ where s has the form $\langle\langle M_1, \ldots, M_t\rangle, c, \mathsf{out}, o\rangle$.

Unwinding Relation. Fixing $s_1^\dagger$ and $d^\dagger$, we construct our $(t * \epsilon^\dagger)$-unwinding relation for a state $s_1^\dagger = \langle\langle M_1^\dagger, \ldots, M_t^\dagger\rangle, c^\dagger, \mathsf{in}, o^\dagger\rangle$ such that $s_1^\dagger \xrightarrow{d^\dagger} \mu^\dagger$. (We know that pc is in since $s_1^\dagger \xrightarrow{d^\dagger} \mu^\dagger$.) Our unwinding relation $\langle \mathcal{U}_1^\cdot, \mathcal{U}_2^\cdot, \beta^\cdot \rangle$ only uses singletons as parts U_1 in $\mathcal{U}_1^\epsilon$ for all ϵ. Likewise, for all ϵ, all U_2 in $\mathcal{U}_2^\epsilon$ is a singleton. We first define $\langle \mathcal{U}_1^\epsilon, \mathcal{U}_2^\epsilon, \beta^\epsilon \rangle$ for $\epsilon = 0$ and then ϵ such that there exists a non-negative integer j such that $\epsilon = j * \epsilon^\dagger$. Lastly, we define it for all other ϵ.

We set $\langle \mathcal{U}_1^{0\epsilon^\dagger}, \mathcal{U}_2^{0\epsilon^\dagger}, \beta^{0\epsilon^\dagger} \rangle$ to be such that $\mathcal{U}_1^{0\epsilon^\dagger}$ is the set of all singletons over S. That is,

$$\mathcal{U}_1^{0\epsilon^\dagger} = \{\, U \subseteq S \mid \exists s \in S \,\text{s.t.}\, U{=}\{s\} \,\}$$

We set $\mathcal{U}_2^{0\epsilon^\dagger} = \mathcal{U}_1^{0\epsilon^\dagger}$ and for $\beta^{0\epsilon^\dagger}$ we use the identity function.

For all integers $j > 0$, we set $\mathcal{U}^{j\epsilon^\dagger}$ to be the set of all singletons $\{s_1\}$ such that s_1 is reachable using only $t - j$ queries and any number of data points. That is, $\{s_1\} \in \mathcal{U}_1^{j\epsilon^\dagger}$ iff there exists $\vec{a} \in A^*$ such that $s_1^\dagger \xrightarrow{\vec{a}} s_1$ and $|\lfloor \vec{a} \rfloor_Q| = t - j$. We set $\mathcal{U}_2^{j\epsilon^\dagger}$ to be those singletons $\{s_2\}$ such that there exists $\vec{a} \in A^*$ such that $\mathsf{add}(s_1^\dagger, d^\dagger) \xrightarrow{\vec{a}} s_2$ and $|\lfloor \vec{a} \rfloor_Q| = t - j$. We set $\beta^{j\epsilon^\dagger}(\{s_1\}) = \{\mathsf{add}(s_1, c^\dagger, d^\dagger)\}$. That is, $\beta^{j\epsilon^\dagger}$ maps states to the state they would have been had they received $d^\dagger$ as input when the `curSlot` was equal to $c^\dagger$.

For all other values of ϵ in $[0, \epsilon^\dagger]$, we define $\langle \mathcal{U}_1^\epsilon, \mathcal{U}_2^\epsilon, \beta^\epsilon \rangle$ to be the same as $\langle \mathcal{U}_1^{j\epsilon^\dagger}, \mathcal{U}_2^{j\epsilon^\dagger}, \beta^{j\epsilon^\dagger} \rangle$ where j is the largest integer such that $\epsilon \geq j * \epsilon^\dagger$. Thus, we round ϵ down to the nearest multiple of $\epsilon^\dagger$.

Lemma 3. *The $(t * \epsilon^\dagger)$-view partition family $\langle \mathcal{U}_1^\cdot, \mathcal{U}_2^\cdot, \beta^\cdot \rangle$ as defined above is an $(t * \epsilon)^\dagger$-unwinding relation.*

Proof. For all ϵ, $s_1 \xrightarrow{a} \mu_1$ iff there exists μ_2 such that $\beta^\epsilon(\{s_1\}) \xrightarrow{a} \mu_2$ since β^ϵ does not change the value of the pc.

Next, we prove that for all ϵ, $\langle \mathcal{U}_1^\epsilon, \mathcal{U}_2^\epsilon, \beta^\epsilon \rangle$ has either Maintenance or Degradation. Let j be the largest integer such that $\epsilon \geq j * \epsilon^\dagger$. We know that that $\langle \mathcal{U}_1^\epsilon, \mathcal{U}_2^\epsilon, \beta^\epsilon \rangle = \langle \mathcal{U}_1^{j\epsilon^\dagger}, \mathcal{U}_2^{j\epsilon^\dagger}, \beta^{j\epsilon^\dagger} \rangle$.

Case: $j = 0$. For all $\{s_1\}$ in $\mathcal{U}_1^0$, $\beta^0(\{s_1\}) = \{s_1\}$ ensuring that Maintenance is trivially satisfied.

Case: $j > 0$. Suppose $U_1 \in \mathcal{U}_1^{j\epsilon^\dagger}$. Since we are always using singletons U_1 is $\{s_1\}$ for some s_1. Furthermore, $\beta^{j\epsilon^\dagger}(U_1)$ has the form $\{\mathsf{add}(s_1, c^\dagger, d^\dagger)\}$. Suppose $s_1 \xrightarrow{a} \mu_1$. In this case, there exists a μ_2 such that $\mathsf{add}(s_1, c^\dagger, d^\dagger) \xrightarrow{a} \mu_2$ by the construction of the system. We now prove either Maintenance or Degradation depending on what type of input a is.

- Subcase: $a \in Q$. Let q be a. In this case, s_1 has the form $\langle\langle M_1, \ldots, M_t\rangle, c, \mathsf{in}, o\rangle$ and μ_1 is such that

$$\mu_1(\mathsf{output}(s_1, o')) = \Pr\left[\kappa_q\left(\biguplus_{c'=1}^{t} M_{c'}\right) = o'\right]$$

  and $\mu_1(s_1') = 0$ for all other states s_1'. $\beta^{j\epsilon^\dagger}(\{s_1\}) = \{s_2\}$ where s_2 is the state $\langle\langle M_1, \ldots, M_{c^\dagger} \uplus \{d^\dagger\}, \ldots, M_t\rangle, c, \mathsf{in}, o\rangle$ and μ_2 is such that

$$\mu_2(\mathsf{output}(s_2, o')) = \Pr\left[\kappa_q\left(\{d^\dagger\} \uplus \biguplus_{c'=1}^{t} M_{c'}\right) = o'\right]$$

and $\mu_2(s_2') = 0$ for all other states s_2'. Since $\biguplus_{c'=1}^{t} M_{c'}$ and $\{d^\dagger\} \uplus \biguplus_{c'=1}^{t} M_{c'}$ differ by one element and κ_q has $\epsilon^\dagger$-differential privacy, we know that for any o',

$$\Pr\left[\kappa_q\left(\biguplus_{c'=1}^{t} M_{c'}\right) = o'\right] \leq \exp(\epsilon^\dagger)\Pr\left[\kappa_q\left(\{d^\dagger\} \uplus \biguplus_{c'=1}^{t} M_{c'}\right) = o'\right]$$

Thus, $\mu_1(\mathsf{output}(s_1, o')) \leq \exp(\epsilon^\dagger)\mu_2(\mathsf{output}(s_2, o'))$ and $\mu_2(\mathsf{output}(s_2, o')) \leq \exp(\epsilon^\dagger)\mu_1(\mathsf{output}(s_1, o'))$. Furthermore, μ_1 and μ_2 assign zero to any states not of this form.

Since $\{s_1\} \in \mathcal{U}^{j\epsilon^\dagger}$, s_1 is reached by $t - j$ queries. Thus, all $s_1' \in \mathsf{Supp}(\mu_1)$ will be reachable in $t - j + 1 = t - (j-1)$ queries and for all $s_1' \in \mathsf{Supp}(\mu_1)$, $\{s_1'\} \in \mathcal{U}_1^{(j-1)*\epsilon^\dagger}$. By similar reasoning, for all $s_2' \in \mathsf{Supp}(\mu_2)$, $\{s_2'\} \in \mathcal{U}_2^{(j-1)*\epsilon^\dagger}$. Thus, to show Degradation, we need only show that for all s_1' reachable in $t - j + 1$ queries, $(j-1) * \epsilon^\dagger \leq \epsilon - \ln(\mu_1(\{s_1'\})) + \ln(\mu_2(\beta^{(j-1)*\epsilon^\dagger}(\{s_1'\})))$ and $(j-1) * \epsilon^\dagger \leq \epsilon + \ln(\mu_1(\{s_1'\})) - \ln(\mu_2(\beta^{(j-1)*\epsilon^\dagger}(\{s_1'\})))$. Since $\epsilon \geq j\epsilon^\dagger$, this is implied by $(j-1) * \epsilon^\dagger \leq j\epsilon^\dagger - \ln(\mu_1(\{s_1'\})) + \ln(\mu_2(\beta^{(j-1)*\epsilon^\dagger}(\{s_1'\})))$ and $(j-1)*\epsilon^\dagger \leq j\epsilon^\dagger + \ln(\mu_1(\{s_1'\})) - \ln(\mu_2(\beta^{(j-1)*\epsilon^\dagger}(\{s_1'\})))$.

To show this, note that we have already shown that $\mu_1(\mathsf{output}(s_1, o')) \leq \exp(\epsilon^\dagger)\mu_2(\mathsf{output}(s_2, o'))$ and $\mu_2(\mathsf{output}(s_2, o')) \leq \exp(\epsilon^\dagger)\mu_1(\mathsf{output}(s_1, o'))$ for all o', and that μ_1 and μ_2 both assign zero to any states not of this form. Thus, we need only show that for all o', $\beta^{(j-1)*\epsilon^\dagger}(\{\mathsf{output}(s_1, o')\}) = \{\mathsf{output}(s_2, o')\}$ where $\mathsf{output}(s_2, o') = \mathsf{output}(\mathsf{add}(s_1, c^\dagger, d^\dagger), o')$. To show this, we consider two subsubcases.

Subsubcase: $j > 1$. In this case, $(j-1) * \epsilon^\dagger > 0$, so $\beta^{(j-1)*\epsilon^\dagger}(\{s_1\}) = \{\mathsf{add}(s_1, c^\dagger, d^\dagger)\}$. Thus, as needed,

$$\begin{aligned}\beta^{(j-1)*\epsilon^\dagger}(\{\mathsf{output}(s_1, o')\}) &= \{\mathsf{add}(\mathsf{output}(s_1, o'), c^\dagger, d^\dagger)\} \\ &= \{\mathsf{output}(\mathsf{add}(s_1, c^\dagger, d^\dagger), o')\} \\ &= \{\mathsf{output}(s_2, o')\}\end{aligned}$$

Subsubcase: $j = 1$. In this case, $t - 1$ queries have been asked by the time s_1 has been reached. Thus, c must be $c^\dagger + t - 1$. Thus, all s_1' reachable $t - (j-1)$ queries, will have the form $\mathsf{output}(s_1, o') = \langle\langle M_1', \ldots, M_t'\rangle, (c^\dagger + t - 1 + 1) \bmod t, \mathsf{out}, o'\rangle$ for some $o' \in O$ where $M'_{(c^\dagger+t-1+1) \bmod t}$ is $\{\!\{\}\!\}$ and $M_{c'}' = M_{c'}$ for all $c' \neq (c^\dagger + t - 1 + 1) \bmod t$. That is, entry $(c^\dagger + t - 1 + 1) \bmod t = c^\dagger$ will be set to the empty multiset. Following the same reasoning for any s_2' reachable in $t - (j-1)$ queries, we find that the entry at $c^\dagger$ will also be set to the empty multiset in s_2'. Since s_1 and s_2 only differ at entry $c^\dagger$, $\mathsf{output}(s_1, o') = \mathsf{output}(s_2, o')$, and thus, $\beta^{(j-1)*\epsilon^\dagger}(\{\mathsf{output}(s_1, o')\}) = \{\mathsf{output}(s_1, o')\} = \{\mathsf{output}(s_2, o')\}$ since $\beta^{(j-1)\epsilon^\dagger} = \beta^0$ is the identity function.

- Subcase: $a \in D$. Let d be a. In this case, s_1 has the form $\langle\langle M_1, \ldots, M_t\rangle, c, \mathsf{in}, o\rangle$ and $\mu_1 = \mathsf{Dirac}(\mathsf{add}(s, c, d))$. Furthermore, $\beta^{j\epsilon^\dagger}(s_1) = \{\mathsf{add}(s_1, c^\dagger, d^\dagger)\}$ and $\mu_2 = \mathsf{Dirac}(\mathsf{add}(\mathsf{add}(s, c^\dagger, d^\dagger), c, d))$. Since it is the case that $\mathsf{add}(\mathsf{add}(s, c^\dagger, d^\dagger), c, d) = \mathsf{add}(\mathsf{add}(s, c, d), c^\dagger, d^\dagger)$,

  $$\beta^{j\epsilon^\dagger}(\{\mathsf{add}(s, c, d)\}) = \{\mathsf{add}(\mathsf{add}(s, c^\dagger, d^\dagger), c, d)\}$$

  So, $\mu_1(\{\mathsf{add}(s, c, d)\}) = 1 = \mu_2(\{\mathsf{add}(\mathsf{add}(s, c^\dagger, d^\dagger), c, d)\})$. Furthermore, $\mu_1(\{s_1'\}) = 0 = \mu_2(\beta^{j\epsilon^\dagger}(\{s_1'\}))$ for all other s_1'. Thus, we have Maintenance.

- Subcase: $a \in O$. Let o be a. In this case, s_1 has the form $\langle\langle M_1, \ldots, M_t\rangle, c, \mathsf{out}, o\rangle$ and $\mu_1 = \mathsf{Dirac}(\mathsf{wait}(s_1))$. Furthermore, $\beta^{j\epsilon^\dagger}(\{s_1\}) = \{\mathsf{add}(s_1, c^\dagger, d^\dagger)\}$ and $\mu_2 = \mathsf{Dirac}(\mathsf{wait}(\mathsf{add}(s_1, c^\dagger, d^\dagger)))$. Since $\mathsf{wait}(\mathsf{add}(s_1, c^\dagger, d^\dagger)) = \mathsf{add}(\mathsf{wait}(s_1), c^\dagger, d^\dagger)$,

  $$\beta^{j\epsilon^\dagger}(\{\mathsf{wait}(s_1)\}) = \mathsf{wait}(\mathsf{add}(s_1, c^\dagger, d^\dagger))$$

  Thus, $\mu_1(\{\mathsf{wait}(s_1)\}) = 1 = \mu_2(\beta^{j\epsilon^\dagger}(\{\mathsf{wait}(s_1)\}))$. Furthermore, $\mu_1(\{s_1'\}) = 0 = \mu_2(\beta^{j\epsilon^\dagger}(\{s_1'\}))$ for all other s_1'. Thus, we have Maintenance.

We have shown either Maintenance or Degradation in all cases. □

Theorem 10. *The system has $(t * \epsilon^\dagger)$-strong trace differential privacy.*

Proof. Since for every $s^\dagger$ such that $s^\dagger \xrightarrow{d^\dagger} \mu^\dagger$, we can define a $(t*\epsilon^\dagger)$-unwinding relation (Lemma 3), we can construct one for all $s^\dagger$ reachable from the initial state s_0. Furthermore, $\mu^\dagger(\beta^{t*\epsilon^\dagger}(\{s_1^\dagger\})) = \mu^\dagger(\{\mathsf{add}(s_1^\dagger, c^\dagger, d^\dagger)\}) = 1$. Thus, Theorem 9 applies the system has $(t * \epsilon^\dagger)$-strong trace differential privacy. □

5 Related Work

Probabilistic models. In defining and reasoning about trace differential privacy, we use tools of concurrency theory. Specifically, our computational model is based on prior work on a form of probabilistic input-output automata model [14]. We formulate trace differential privacy to capture the idea that an automaton implementing a differentially private function produces approximately the same distribution when fed with inputs that differ in only one data point. In order to prove such a property about a given automaton, we define an unwinding relation over automata that ensures that the desired privacy error bound is achieved by keeping track of the error introduced in every transition of the automaton. One particularly related work is by Segala and Turrini, who have studied reasoning techniques based on approximate simulation relations in the context of cryptographic protocols [22]. The approximation in that work stems from the allowable probability that certain transitions of the protocol do not have a matching transition in the specification. This models the capability of the adversary to compromise correctness. A protocol is deemed correct if the error accumulated at the end of a polynomial length execution is exponentially small in some security parameter. Our unwinding technique, on the other hand, requires that there always be a matching transition (if a response is possible from one state it should always be possible from a related state). However, the probabilities of those transitions are within some exponential multiplicative factor of one another. In this sense, neither approach subsumes the other.

Noninterference. In formalizing differential privacy, we found some surprising similarities with noninterference. Noninterference separates users into low and high levels and requires that a low-level user learns nothing about the inputs of a high-level user by interacting with the system [10]. This property was originally defined for deterministic systems, but has been extended to probabilistic systems [12, 13]. Probabilistic noninterference requires the probabilities of traces of the system observable by low-level users to be identical for any pair of high-level inputs (data points in our setting). To weaken this definition, Di Pierro, Hankin, and Wiklicky introduced *approximate non-interference* [21] and Backes and Pfitzmann introduced *computational probabilistic non-interference* [2]. While these properties allow for the probabilities to be close (without being equal), they both do not allow the systems to diverge as the difference between the high-level inputs increase. In contrast, weak trace differential privacy allows for the system to increasingly diverge as the high-level inputs (the data points) diverge. This is needed in order to release meaningful statistics.

Quantitative Information Flow Analysis. There is also some similarity with prior work on quantitative information flow analysis. Quantitative information flow analysis attempts to determine how much information a program provides an adversary about a sensitive input or class of inputs. Clark, Hunt, and Malacaria present a formal model of programs for quantifying information flows and a static analysis that provides lower and upper bounds on the amount of information that flows [5]. They measure information flow as the mutual information between the high-level inputs and low-level outputs given that the adversary has control over the low-level inputs. Malacaria extends this work to handle loops [15], and Chen and Malacaria to multi-threaded programs [4]. McCamant and Ernst [16], and Newsome and Song [19] provide dynamic analyses for quantitative information flow using the mutual information formalization. All of the above approaches assume that the adversary's beliefs are aligned with the actual distribution producing the sensitive input(s) and that adversary has no additional background knowledge. Clarkson, Myers, and Schneider object to the mutual information formulation of quantitative information flow [6]. Instead they propose

a formulation using the beliefs of the adversary. However, such a formulation may be difficult to apply in practice because the surveyor may not know the adversary's beliefs. An advantage of differential privacy is that no assumptions are needed about the adversary's auxiliary information, computational power or beliefs. However, it only provides a relative privacy guarantee and not an absolute quantitative information flow guarantee like other work mentioned above.

6 Future Work

In this paper, we made substantial progress towards developing a basis of differential privacy for systems. We plan to extend this work in several directions. First, a general characterization of conditions on query schedulers, data sets, and differentially private functions that allows trace differential privacy to be achieved in an asynchronous setting is an interesting open problem. Second, we plan to carry out a detailed case study of the PINQ system design to evaluate both the robustness of our definitions and the guarantees that PINQ provides. Finally, we plan to extend the theory to model and reason about higher level systems that use a differentially private database system as a component. Examples of such systems include processes in organizations such as hospitals that release privacy-preserving aggregate information as well as distributed systems that use differential privacy as a basis for releasing privacy-preserving information [1]. We envision that this line of work will lead to a useful theoretical basis for reasoning about the privacy guarantees provided by practical systems that release aggregate information about a data set containing sensitive information about individuals.

References

[1] Anonymized. Airavat: Security and privacy for MapReduce. Authors redacted due to blind reviewing process., 2009.

[2] Backes, M., and Pfitzmann, B. Computational probabilistic non-interference. In *ESORICS '02: Proceedings of the 7th European Symposium on Research in Computer Security* (London, UK, 2002), Springer-Verlag, pp. 1–23.

[3] Blumn, A., Ligett, K., and Roth, A. A learning theory approach to non-interactive database privacy. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing* (New York, NY, USA, 2008), ACM, pp. 609–618.

[4] Chen, H., and Malacaria, P. Quantitative analysis of leakage for multi-threaded programs. In *PLAS '07: Proceedings of the 2007 workshop on Programming languages and analysis for security* (New York, NY, USA, 2007), ACM, pp. 31–40.

[5] Clark, D., Hunt, S., and Malacaria, P. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security 15* (2007), 321–371.

[6] Clarkson, M. R., Myers, A. C., and Schneider, F. B. Belief in information flow. In *CSFW '05: Proceedings of the 18th IEEE workshop on Computer Security Foundations* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 31–45.

[7] Dwork, C. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)* (2006), vol. 2, pp. 1–12.

[8] Dwork, C. *Theory and Applications of Models of Computation*, vol. 4978. Springer, 2008, ch. Differential Privacy: A Survey of Results, pp. 1–19.

[9] Dwork, C. The differential privacy frontier (extended abstract). In *6th Theory of Cryptography Conference* (2009), vol. 5444 of *Lecture Notes in Computer Science*, Springer, pp. 496–502.

[10] Goguen, J. A., and Meseguer, J. Security policies and security models. In *IEEE Symposium on Security and Privacy* (1982), IEEE, p. 11.

[11] Goguen, J. A., and Meseguer, J. Unwinding and inference control. In *Proc. of IEEE Symp. on Security and Privacy* (Los Alamitos, CA, USA, 1984), IEEE Computer Society, pp. 75–86.

[12] Gray, III, J. W. Toward a mathematical foundation for information flow security. In *IEEE Symposium on Security and Privacy* (1991), pp. 21–35.

[13] Gray, III, J. W. Toward a mathematical foundation for information. *Journal of Computer Security 1*, 3-4 (1992), 255–294.

[14] Lynch, N., Segala, R., and Vaandrager, F. Observing branching structure through probabilistic contexts. *SIAM Journal on Computing 37*, 4 (2007), 977–1013.

[15] Malacaria, P. Assessing security threats of looping constructs. In *POPL '07: Proceedings of the 34th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (New York, NY, USA, 2007), ACM, pp. 225–235.

[16] McCamant, S., and Ernst, M. D. A simulation-based proof technique for dynamic information flow. In *PLAS '07: Proceedings of the 2007 workshop on Programming languages and analysis for security* (New York, NY, USA, 2007), ACM, pp. 41–46.

[17] McSherry, F. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *SIGMOD '09: Proceedings of the 2009 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2009), ACM. To appear. Available at `http://research.microsoft.com/apps/pubs/?id=80218`.

[18] McSherry, F., and Talwar, K. Mechanism design via differential privacy. In *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 94–103.

[19] Newsome, J., and Song, D. Influence: A quantitative approach for data integrity. Tech. Rep. CMU-CyLab-08-005, CyLab, Carnegie Mellon University, 2008.

[20] Nissim, K., Raskhodnikova, S., and Smith, A. Smooth sensitivity and sampling in private data analysis. In *STOC '07: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing* (New York, NY, USA, 2007), ACM, pp. 75–84.

[21] Pierro, A. D., Hankin, C., and Wiklicky, H. Approximate non-interference. *J. Comput. Secur. 12*, 1 (2004), 37–81.

[22] Segala, R., and Turrini, A. Approximated computationally bounded simulation relations for probabilistic automata. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium* (Venice, Italy, 2007), pp. 140–156.