Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science

Electronic Notes in Theoretical Computer Science 276 (2011) 61–79

www.elsevier.com/locate/entcs

Formal Verification of Differential Privacy for Interactive Systems

Extended Abstract [1]

Michael Carl Tschantz, Dilsun Kaynar, and Anupam Datta[2]

Carnegie Mellon University
Pittsburgh, USA

Abstract

Differential privacy is a promising approach to privacy preserving data analysis with a well-developed theory for functions. Despite recent work on implementing systems that aim to provide differential privacy, the problem of formally verifying that these systems have differential privacy has not been adequately addressed. We develop a formal probabilistic automaton model of differential privacy for systems by adapting prior work on differential privacy for functions. We present the first sound verification technique for proving differential privacy of interactive systems. The technique is based on a form of probabilistic bisimulation relation. The novelty lies in the way we track quantitative privacy leakage bounds using a relation family instead of a single relation. We illustrate the proof technique on a representative automaton motivated by PINQ, an implemented system that is intended to provide differential privacy. Surprisingly, our analysis yields a privacy leakage bound of $(2t * \epsilon)$ rather than $(t * \epsilon)$ when ϵ-differentially private functions are called t times. The extra leakage arises from accounting for bounded memory constraints of real computers.

Keywords: Differential Privacy, Verification, Formal Methods, Privacy, Bisimulation

1 Introduction

Differential Privacy. Differential privacy is a promising approach to privacy-preserving data analysis (see [14,16] for surveys). 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

[1] 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 Carnegie Mellon CyLab, the NSF Science and Technology Center TRUST, the NSF CyberTrust grant "Privacy, Compliance and Information Risk in Complex Organizational Processes," and the AFOSR MURI "Collaborative Policies and Assured Information Sharing."

[2] Email: mtschant@cs.cmu.edu, dilsunk@cmu.edu, danupam@cmu.edu

doi:10.1016/j.entcs.2011.09.015

the underlying population while simultaneously protecting the privacy of the individuals. In an interactive setting, an untrusted data examiner poses queries that the trusted party evaluates over the data set and appropriately modifies to protect privacy before sending the result to the examiner.

Differential privacy formalizes this operation in terms of a probabilistic *sanitization function* that takes the data set as input. Differential privacy requires that the probability of producing an output should not change much irrespective of whether information about any particular individual is in the data set or not. The amount of change is measured in terms of a *privacy leakage bound*—a non-negative real number ϵ, where a smaller ϵ indicates a higher level of privacy. 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 societal benefits from doing so (e.g., aiding 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. Previous work on algorithms for sanitization functions and the analysis of these algorithms in light of the trade-offs between privacy and utility (answering useful queries accurately without compromising privacy) has provided firm foundations for differential privacy (e.g. [17,13,30,34,6,14,15,20,16,18]).

In a different direction, these sanitization algorithms are being implemented for inclusion in data management systems. For example, PINQ resembles a SQL database, but instead of providing the actual answer to SQL queries, it provides the output of a differentially private sanitization function operating on the actual answer [29]. Another such system, AIRAVAT, manages distributed data and performs MapReduce computations in a cloud computing environment while using differential privacy as a basis for declassifying data in a mandatory access control framework [39]. Both of these are interactive systems that use sanitization functions as a component: they interact with both the providers of sensitive data and untrusted data examiners, store the data, and perform computations on the data some of which apply sanitization functions.

Even if we assume that these systems correctly implement the sanitization functions to give differential privacy, this is not sufficient to conclude that the guarantees of differential privacy apply to the system as a whole. For the differential privacy guarantee of functions to scale to the whole of the implemented system, the system must properly handle the sensitive data and never provide channels through which untrusted examiners can infer information about it without first sanitizing it to the degree dictated by the privacy error bound.

Formal Methods for Differential Privacy. We work toward reconciling formal analysis techniques with the growing body of work on abstract frameworks or implemented systems that use differential privacy as a building block. While prior work in the area has provided a type system for proving that a non-interactive program is a differentially private sanitization function [38], we know of no formal methods that can, in addition, prove that an interactive system using such functions has differential privacy. Applying formal methods to interactive systems ensures

that these systems properly manage their data bases and interactions with untrusted users.

We work with a special class of probabilistic I/O automata that allow us to model interactive systems in terms of states and probabilistic transitions between states. These automata provide us with the needed expressive power for modeling how data is stored in an internal state of an implementation, and how it is updated through computations, some of which apply differentially private sanitization functions on data. We present this probabilistic automaton model and our differential privacy definition for probabilistic automata, which we call *differential noninterference* due to the similarities it has with the information flow property *noninterference* [21]. Indeed, when applied to interactive systems, both differential privacy and noninterference privacy aim at restricting information leakage about sensitive data by requiring that the system produces similar outputs for inputs that differ only in sensitive data. However, differential privacy allows for the degree of similarity to decrease as the inputs diverge, making it a more flexible requirement.

Our main technical contribution is a *proof technique* for establishing differential non-interference guarantees for programs implementing sanitization functions (such as the Truncated Geometric Mechanism [20]) and of interactive systems that use such functions as building blocks. Our technique allows the global property of differential noninterference to be proved from local information about transitions between states. This proof technique was inspired by the *unwinding* proof technique originally developed for proving that a system has noninterference [22].

Our unwinding technique is also similar to bisimulation-based proof techniques as both uses a notion of "similarity" of states with respect to their observable behavior. Unlike traditional bisimulation relations for probabilistic automata, the unwinding relation is defined over the states of a single automaton with the intention of establishing the similarity of two states where one is obtainable from the other by the input of an additional data point. Moreover, the notion of similarity is approximate, which is in keeping with the definition of differential privacy. An unwinding proof involves finding a relation family indexed by the set of possible values of the privacy leakage bound ϵ, rather than a single relation. This departure from traditional probabilistic bisimulations is needed to track the maximum privacy leakage tolerable from a given state in the execution. We prove the soundness of our proof technique: the existence of appropriate ϵ-unwinding families for an automaton M implies that M has ϵ-differential noninterference.

We illustrate the proof technique on a representative automaton motivated by PINQ, an implemented system that is intended to provide differential privacy. Surprisingly, our analysis yields a privacy leakage bound of $(2t * \epsilon)$ rather than $(t * \epsilon)$ when ϵ-differentially private functions are called t times. The extra leakage arises from accounting for bounded memory constraints of real computers.

An earlier version of this work [44], in addition to the unwinding proof technique, considers asynchronous systems in which queries are not necessarily answered in the order in which they are posed and demonstrates that this additional flexibility has subtle consequences for differential privacy. Further details of this line of work,

including definitions and results related to compositional reasoning and automation of our proof technique can be found in our related technical report [45].

2 Background

2.1 Differential Privacy

Differential privacy formalizes the idea that a private process should not reveal too much information about a single person. A *data point* represents all the information collected about an individual (or other entity that must be protected). A multiset (bag) of data points forms a *data set*. A *sanitization function* κ processes the data set and returns a result to the untrusted data examiner that should probabilistically not change whether or not a single data point is in the data set. Dwork [13] states differential privacy as follows:

Definition 2.1 [Differential Privacy] A randomized function κ has 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 $B_2 \cup \{d\} = B_1$.

Privacy Mechanisms. As shown in the original work on differential privacy, given a statistic f that can be computed of the data sets B_i, one can construct a sanitization function κ_f from f by having κ_f add noise to the value of $f(B_i)$ where the noise is drawn from a Laplace distribution [17]. This is an example of a *privacy mechanism*, a scheme for converting a statistic into a sanitization function with differential privacy.

Systems in practice would implement a sanitization function such as κ_f as a program. As actual computers have only a bounded amount of memory, the program computing κ_f must only use a bounded amount of memory. However, many sanitization functions proposed in the differential privacy literature, including all sanitization functions constructed using the Laplace privacy mechanism, use randomly drawn real numbers, which requires an uncountably infinite number of states. While such functions can be approximated using a finite number of states (e.g., by using floating point numbers), it is unclear whether the proofs that these functions have differential privacy carry over to their approximations.

As we are interested in formally proving that finite systems provide differential privacy, we limit ourselves to privacy mechanisms that operate over only a finite number of values. One such mechanism is the *Truncated Geometric Mechanism* of Ghosh et al. [20], which uses noise drawn from a bounded, discrete version of the Laplace distribution.

```
01 dPts:= emptyArray(t);
02 numPts := emptyArray(t);
03 for(j:=0; j<t; j++)
04   dPts[j]:= emptyArray(maxPts);
05   numPts[j] := 0;
06 curSlot:=0;
07 while(1)
08   y:=input();
09   if(datapoint(y))
10     if(numPts[curSlot]<maxPts)
11       dPts[curSlot][numPts[curSlot]]:=y;
12       numPts[curSlot]++;
13   else
14     k:=get_sanitization_funct(y);
15     res:=k.compute(dPts);
16     print(res);
17     curSlot:=(curSlot + 1) mod t;
18     delete dPts[curSlot];
19     dPts[curSlot] := emptyArray(maxPts);
20     numPts[curSlot] := 0
```

Fig. 1. Program that tracks data point usage to ensure differential noninterference

2.2 Motivating Example System

To further motivate and illustrate our work, we provide an example of an interactive system that uses sanitization functions. The system manages data points entered by data providers and processes requests of data examiners for information by receiving queries and answering them after sanitizing the answer computed over the data set. The system must apply the sanitization functions to the data set and interact with the data examiner in a manner that does not compromise privacy.

Possible source code for one such system is shown in Figure 1. To be concrete, suppose that the data points are integers and the system handles only two queries. The first produces the output of the sanitization function COUNT, which provides the number of data points currently in the data base. The second produces the output of SUM, which provides their sum. In both cases, the sanitization functions use the Truncated Geometric Mechanism to preserve privacy [20].

Intuitively, the program in Figure 1 keeps an array of t arrays of data points and a variable `curSlot`, whose value indicates a (current) slot in the array. If the input is a data point, that data point is added to the array indexed by `curSlot` unless that array is full, in which case the data point is ignored.

If the input is a query, then the query requested by the input is computed on the union of all the data points collected from all the arrays. Line `15` uses either the implementation of COUNT or SUM to compute the system's response to the query `y` where Line `14` selects the correct function. Furthermore, the index `curSlot` to one of these arrays is cyclically shifted and the array to which it now points is replaced

with an empty array. Since there are only t slots, this means that each array will only last for t queries before being deleted. (If $t = 0$, we take the program to have an array `dPts` of length 0, in which case it never stores any data points.) Since each query has ϵ-differential privacy, this ensures that each data point will only be involved in $t * \epsilon$ worth of queries.

Verification. The goal of our work is to formally verify that systems like this one preserve the privacy of their users. In addition to showing that the sanitization functions COUNT and SUM have differential privacy (a subject of previous work [20]), we study how the system leaks information about the data points in ways other than through the outputs from these functions. Indeed, one might expect from the sequential result for differential privacy discussed above [30, Corollary 5], that the system would provide $(t * \epsilon)$-differential privacy. However, due to how the system manages data points, it actually only provides $(2t*\epsilon)$-differential privacy as we show later.

Had our goal only been to formally verify the implementations of the sanitization functions COUNT and SUM, it would suffice to use a simple formal model such as that of probabilistic finite-state automata with no interaction and use a suitable algorithmic technique to verify differential privacy, which research on Markov chains provides.

However, to verify differential privacy for interactive systems that use privacy mechanism as a building block as the above system does, we need a more expressive formal model that models the interaction of the data examiner with the system and the addition of data points to the system over time. The next section provides such a model.

3 Modeling Interaction for Formal Verification

In this section, we present the basics of the formal framework we use in modeling interactive systems and show how we can model the example system of Section 2.2 using this formalism. Specifically, in Sections 3.1 and 3.2, we introduce a special class of probabilistic I/O automata and present our definition of differential privacy for this class of probabilistic I/O automata. In Section 3.3 we model the program of Figure 1 as a probabilistic I/O automaton.

3.1 Automata

We use a simplified version of probabilistic I/O automata (cf. [25]). We define an automaton in terms of a probabilistic labeled transition system (PLTS).

Definition 3.1 A probabilistic labeled transition system (PLTS) is a tuple $L = \langle S, I, O, \rightarrow \rangle$ where S is a countable set of states; I and O are countable and pairwise disjoint sets of actions, referred to as *input* and *output* actions respectively; and $\rightarrow \subseteq S \times (I \cup O) \times \mathsf{Disc}(S)$ represents the possible transitions where $\mathsf{Disc}(S)$ is the set of discrete probability measures over S.

We use A for $I \cup O$. We partition the input set I into D, the set of data points, and Q, the set of queries. We also partition the output set O into R, the set of responses to the data examiner's queries and H, the set of outputs that are hidden from (not observable to) the data examiner. Note that H includes outputs to the data provider. We let E range over all actions to which the examiner has direct access: $E = Q \cup R$. When only one automaton is under consideration, we denote a transition $\langle s, a, \mu \rangle \in \rightarrow$ by $s \xrightarrow{a} \mu$.

Henceforth, we require that PLTSs 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. We require that each output producing state produces only one output since the choice of output should be made by the PLTS to avoid nondeterminism that might be resolved in a way that leaks information about the data set. Owing to transition determinism, we will often write $s \xrightarrow{a} \mu$ without explicitly quantifying μ.

We define an extended transition relation $\Rightarrow$ that describes how a PLTS may perform a sequence of actions where some of the output actions are hidden from the data examiner. In particular, the hidden outputs in H model unobservable internal actions irrelevant to privacy. To define $\Rightarrow$, let a state that produces an output from H be called *H-enabled* and one that does not be called *H-disabled*. By output determinism, H-enabled states may only transition under an action in H and, thus, cannot have transitions on actions from $R \cup Q \cup D$. To skip over such states and focus on H-disabled states, which are more interesting from a verification point of view, we define $\Rightarrow$ to show to which H-disabled states the system may transition while performing any finite number of hidden actions. We define $s \overset{a}{\Rightarrow} \nu$ so that $\nu(s')$ is the probability of reaching the H-disabled state s' from the state s where a is the action performed from state s. Note that ν is not a distribution over the set S of states since the automaton might execute an infinite sequence of H-enabled states never reaching an H-disabled state. We let ν be a distribution over $S_\perp = S \cup \{\perp\}$ where $\perp \notin S$ represents nontermination and $\nu(\perp) = 1 - \sum_{s \in S} \nu(s)$. Note that for no a, μ, or ν does $\perp \xrightarrow{a} \mu$ or $\perp \overset{a}{\Rightarrow} \nu$.

A PLTS L combined with a state s defines a probabilistic I/O automaton $\langle L, s \rangle$. This state is thought of as the initial state of the automaton or the current state of the PLTS. We define a *trace* to be a sequence of actions from $A^* \cup A^\omega$. Given such an automaton M, we define $[\![M]\!]$ to be a function from input sequences to the random variable over traces that describes how the automaton M behaves under the inputs $\boldsymbol{i}$. We let $\lfloor [\![M]\!](\boldsymbol{i}) \rfloor_E$ denote the random variable over sequences of actions

observable to the data examiner obtained by projecting only the actions in E from the trace returned by random variable $[\![M]\!](\boldsymbol{i})$.

To deal with nontermination, we note that the examiner can only observe finite prefixes of any nonterminating trace. When the examiner sees the finite prefixes of a trace, he must consider all traces of the system with the observed prefix as possible. (The set of these traces has been called a *cone* — see e.g. [25].) Since the examiner may only see actions in E, these sets are in one-to-one correspondence with E^*. Thus, the examiner observing some event is not modeled as the probability of the system producing a trace in some set, but rather with the probability of a system producing a prefix of trace in some set. That is, rather than using $\Pr[\lfloor [\![M]\!](\boldsymbol{i})\rfloor_E \in S]$ for $S \subseteq E^* \cup E^\omega$, we need $\Pr[\lfloor [\![M]\!](\boldsymbol{i})\rfloor_E \sqsupseteq S]$ for $S \subseteq E^*$ where $\sqsupseteq$ is the super-sequence-equal operator raised to work over sets in the following manner: $\boldsymbol{e} \sqsupseteq S$ iff there exists $\boldsymbol{e}' \in S$ such that $\boldsymbol{e} \sqsupseteq \boldsymbol{e}'$ where $\boldsymbol{e} \in E^* \cup E^\omega$ and $S \subseteq E^*$.

3.2 Differential Noninterference

Often the data set of a differentially private system is loaded over time and may change between queries. Such changes in the data set are not explicitly modeled by the definition of differential privacy, but one could conceive of modeling such changes by having data points be time-indexed sequences of data. Nevertheless, for formal verification, we require an explicit model of data set mutation. Thus, we present a version of differential privacy defined in terms of the behavior of an automaton that accepts both queries and data points over time.

Definition 3.2 [Differential Noninterference] An automaton M has ϵ-differential noninterference if for all input sequences $\boldsymbol{i}_1$ and $\boldsymbol{i}_2$ in I^* differing on at most one data point, and for all $S \subseteq E^*$,

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

where we say two input sequences differ by one data point if one of the sequences may be constructed from the other by inserting a single data point anywhere in it.

By restricting the traces of M to only those elements of $E = Q \cup R$, 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 $\boldsymbol{i}_1$ and $\boldsymbol{i}_2$, which differ by at most one data point. Note that like the original form of differential privacy, we do not model the adversary explicitly but rather consider the behavior of the automaton over all possible input sequences the adversary could supply.

3.3 Example: Automaton Model for Program of Figure 1

To eventually prove that the program of Figure 1 has $(2t * \epsilon)$-differential noninterference, we first give an automaton model of the program, called $M_{\mathsf{ex1}}(K)$. Note that the model we give here is parametric in the set of sanitization functions; it applies not only to the program of Figure 1, which assumes $K = \{\textsc{count}, \textsc{sum}\}$

but to any other instance of the same program that uses a possibly different set of sanitization functions (modeled by the parameter K). We define below the state space S and transition relation $\rightarrow$, which determine $L_{\mathsf{ex1}}(K) = \langle S, I, O, \rightarrow\rangle$ for every set K of sanitization functions. Using an initial state s_0, we get the automaton $M_{\mathsf{ex1}}(K) = \langle L_{\mathsf{ex1}}(K), s_0\rangle$.

States. Each state of the automaton can be viewed as a particular valuation of the variables in the program allowed by its type. We model the array `dPts` as a t-tuple of multisets of data points. We model `numPts` as a t-tuple of integers ranging from 0 to v where v is the value held by the constant `maxPts`. We model the index `curSlot` as an integer c ranging from 0 to $t-1$, which selects one of the multisets of the t-tuple. The variable `y` stores the most recent input. The variable `res` keeps track of which output from O is about to be produced and the sanitization function is stored in `k`. The state must also keep track of a program counter pc, which ranges over the program line numbers from 01 to 20. Thus, the set of states S is $\{01,\ldots,20\} \times (\mathsf{bag}(D))^t \times \{0,\ldots,v\}^t \times \{0,\ldots,t-1\} \times I \times O \times K$ where $\mathsf{bag}(D)$ is the set of all multisets with elements from D and K is the set of sanitization functions.

Actions. We model the `input` command in the source code with the input action set I of our automaton: for each possible value that `input` can return there is an input action in I corresponding to that value. Inputs in the code can be either queries or data points, which is modeled by the partition of the set I into the sets Q for queries and D for data points. We model the `print` command in the source code with the observable outputs R (responses) of our automaton. For each possible value that can be printed we have an output action in R. We model all other commands by internal (hidden) actions.

Transitions. We list below only those transitions that are interesting for our purposes. That is, transitions on actions from the sets I and R, and transitions on hidden actions that represent internal computation such as choosing of an appropriate sanitization function for a given query and computation of the result using that function. We use the symbol τ for hidden actions. We also use *Dirac* distributions: let $\mathsf{Dirac}(s)$ be the distribution such that $\Pr[\mathsf{Dirac}(s){=}s] = 1$ and $\Pr[\mathsf{Dirac}(s){=}s'] = 0$ for all $s' \neq s$. Given a query q in Q, we let κ_q be the sanitization function that answers that query. Some key transitions are:

Input $\langle 08, \boldsymbol{B}, \boldsymbol{n}, c, y, r, k\rangle \xrightarrow{i} \mathsf{Dirac}(\langle 09, \boldsymbol{B}, \boldsymbol{n}, c, i, r, k\rangle)$

Choose Function $\langle 14, \boldsymbol{B}, \boldsymbol{n}, c, y, r, k\rangle \xrightarrow{\tau} \mathsf{Dirac}(\langle 15, \boldsymbol{B}, \boldsymbol{n}, c, y, r, \kappa_y\rangle)$

Compute Function $\langle 15, \langle B_0, \ldots, B_{t-1}\rangle, \boldsymbol{n}\rangle, c, y, r, k\rangle \xrightarrow{\tau} \mu$ where

$$\mu(\langle 16, \langle B_0, \ldots, B_{t-1}\rangle, \boldsymbol{n}, c, y, r', k\rangle) = \Pr[k(\biguplus_{\ell=0}^{t-1} B_\ell) = r']$$

using $\uplus$ for multiset union and $\mu(s') = 0$ for states not of that form, and

Output Result $\langle 16, \boldsymbol{B}, \boldsymbol{n}, c, y, r, k\rangle \xrightarrow{r} \mathsf{Dirac}(\langle 17, \boldsymbol{B}, \boldsymbol{n}, c, y, r, k\rangle)$

The third transition above is a probabilistic transition that represents the internal computation of a sanitization function k on the union of the multisets $B_0, \ldots, B_{t-1}$. The effect of the transition is to update the value of the pc from 15 to 16 and to update the result to be output from r to a new value r' such that the probability of ending up in state $\langle 16, \langle B_1, \ldots, B_t\rangle, c, n, y, r', k\rangle$ as a result of the transition is $\Pr[k(\biguplus_{\ell=1}^{t} B_\ell) = r']$.

From these transitions, we can calculate the extended transitions for each of the three types of H-disabled states:

Drop $\langle 08, \boldsymbol{B}, \boldsymbol{n}, c, y, r, k\rangle \stackrel{d}{\Rightarrow} \mathsf{Dirac}(\langle 08, \boldsymbol{B}, \boldsymbol{n}, c, d, r, k\rangle)$ when n_c of $\boldsymbol{n}$ is v;

Add $\langle 08, \boldsymbol{B}, \boldsymbol{n}, c, y, r, k\rangle \stackrel{d}{\Rightarrow} \mathsf{Dirac}(\langle 08, \boldsymbol{B'}, \boldsymbol{n'}, c, d, r, k\rangle)$ when n_c of $\boldsymbol{n}$ is less than v and $\boldsymbol{B'}$ and $\boldsymbol{n'}$ are such that $B'_c = B_c \uplus \{d\}$, $n'_c = n_c + 1$, and for all $c' \neq c$, $B'_{c'} = B_{c'}$ and $n'_{c'} = n_{c'}$;

Answer Query $\langle 08, \langle B_0, \ldots, B_{t-1}\rangle, \boldsymbol{n}, c, y, r, k\rangle \stackrel{q}{\Rightarrow} \nu$ where

$$\nu(\langle 16, \langle B_0, \ldots, B_{t-1}\rangle, \boldsymbol{n}, c, q, r', \kappa_q\rangle) = \Pr[k(\biguplus_{\ell=0}^{t-1} B_\ell) = r']$$

and $\nu(s') = 0$ for states not of that form; and

Delete Old Data $\langle 16, \boldsymbol{B}, \boldsymbol{n}, c, y, r, k\rangle \stackrel{r}{\Rightarrow} \mathsf{Dirac}(\langle 08, \boldsymbol{B'}, \boldsymbol{n}, c, y, r, k\rangle)$ where we have $B'_{c+1 \mod t} = \{\!\{\}\!\}$, $n'_{c+1 \mod t} = 0$, and for all $c'' \neq c+1 \mod t$, $B'_{c''} = B_{c''}$ and $n'_{c''} = n_{c''}$ using $\{\!\{\}\!\}$ for the empty multiset.

The third extended transition above represents a sequence of transitions that starts with the input of a query q. The input of the query is followed by transitions on hidden actions that model the computation of the answer to the query where some of these hidden steps are probabilistic. The resulting state has the property that κ_q has been chosen as the sanitization function and that $pc = 16$, which implies that the resulting state is H-disabled and the automaton is ready to perform an observable output by outputing the answer to the query.

The state space S and transition relation $\rightarrow$ determines the PLTS $L_{\mathsf{ex1}}(K) = \langle S, I, O, \rightarrow\rangle$ for every set K of differentially private functions. Using the initial state $s_0 = \langle 1, \{\!\{\}\!\}^t, 0^t, 1, y_0, r_0, k_0\rangle$, we get the automaton $M_{\mathsf{ex1}}(K) = \langle L_{\mathsf{ex1}}(K), s_0\rangle$. (The initial values y_0, r_0, k_0 do not matter since they will be replaced before being used.)

4 Unwinding Proof Technique

We desire a technique for drawing conclusions about the global behavior (executions) of the system from local aspects (states, actions, and transitions) of the model. Faced with a similar situation, Goguen and Meseguer introduced unwinding relations to simplify proving that a system has noninterference [22]. We present a similar technique for proving that a system has differential noninterference. In particular we state what it means for a relation family to be an unwinding family

and prove Theorem 4.5, which roughly states that the existence of an unwinding family for a given automaton implies that it satisfies differential noninterference. Our unwinding notion is probabilistic and approximate, which is in keeping with the notion of differential privacy. The novelty lies in the way we keep track of the privacy leakage bound, which evolves as the system evolves where the evolution is constrained by the differential privacy definition.

4.1 Definition and Soundness

Formulating a notion of unwinding relation that is sound for showing differential noninterference is more complicated than existing notions for showing noninterference because we must deal with probabilities and we must keep track of the privacy leakage bound ϵ. To deal with probabilities and approximation, we adapt the notion of *approximate lifting* from previous work on approximate probabilistic simulation relations in the context of cryptographic protocols [42]. However, such work does not deal with tracking a leakage bound (see Section 5 for additional details). Thus, we introduce a *family* of unwinding relations indexed by various amounts of privacy leakage. Each unwinding relation in the family is a relation on the state space of the automaton. The unwinding relation indexed by the leakage amount ϵ relates states that exhibit approximately the same trace distributions in the sense of ϵ-differential noninterference.

To deal with probabilities in a concise and modular way, we first define an approximate lifting operation that takes a relation over sets and produces a relation over distributions on those sets. The degree of approximation is governed by a parameter δ.

Definition 4.1 [δ-Approximate Lifting] Let R be a relation between a set X and a set Y. The *δ-approximate lifting* of R denoted by $\mathcal{L}(\mathsf{R}, \delta)$ is the relation between $\mathsf{Disc}(X)$ and $\mathsf{Disc}(Y)$ such that for all ν_1 in $\mathsf{Disc}(X)$ and ν_2 in $\mathsf{Disc}(Y)$, $\nu_1 \; \mathcal{L}(\mathsf{R}, \delta) \; \nu_2$ if and only if there exists a bijection $\beta : \mathsf{Supp}(\nu_1) \to \mathsf{Supp}(\nu_2)$ such that for all x in $\mathsf{Supp}(\nu_1)$, $x \; \mathsf{R} \; \beta(x)$ and $|\ln \nu_1(x) - \ln \nu_2(\beta(x))| \leq \delta$.

The requirement for β to be from the support set of ν_1 to the support set of ν_2 ensures that if a state is assigned a non-zero probability in ν_1 then it is not possible for a related state to be assigned a zero probability in ν_2 and vice versa—there is one to one correspondence between the states with non-zero and identical probabilities in the two distributions. The form of δ involves natural logarithms because the privacy leakage bound in the differential privacy definition appears in the exponent.

Next we define our unwinding technique, which is illustrated in Figure 2. Intuitively, since we want the behavior of the automaton to change only by a factor of ϵ on receiving a single data point, we want the transitions under a data point from a state s to lead to states s' that are only a factor of ϵ different from s. *Covering* (Definition 4.4) formalizes this by requiring that state s is related to each such state s' by a relation $\mathcal{R}^\epsilon$ that is part of an *ϵ-unwinding family* (Definition 4.2).

In more detail, an ϵ-unwinding family starts with a privacy leakage budget of ϵ, which decreases over time to a current balance of ϵ'. Related states s_1 and s_2 are

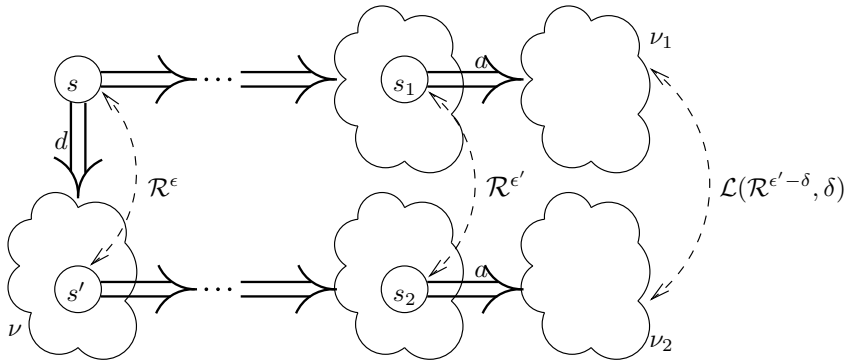

Fig. 2. Unwinding Family and Covering: The left side shows the requirements for a covering. The right side shows the requirements placed on an unwinding family. The solid arrows denote the extended transition relation $\Rightarrow$ and clouds depict probability distributions such as ν where $s' \in \mathsf{Supp}(\nu)$.

required to only make transitions under the same actions. The distributions ν_1 and ν_2 that result from these transitions followed by any number of transitions under hidden outputs may differ only by a factor of δ. This difference is subtracted from the current balance ϵ' to get a new current balance. Once the balance reaches zero, the resulting distributions must be equivalent. As the balance started at ϵ, only a total of ϵ privacy can be leaked, a point proved in Lemma 4.3.

Definition 4.2 [ϵ-Unwinding Family] For a non-negative real number ϵ, a family indexed by the set $[0, \epsilon]$ of relations $\mathcal{R}^{\cdot}$ over the H-disabled states of a PLTS L is an *ϵ-unwinding family* for L if for all ϵ' in $[0, \epsilon]$, for all x_1 and x_2 in $S_\perp$ such that $x_1 \; \mathcal{R}^{\epsilon'} \; x_2$, for all a in $I \cup R$, there exists ν_1 such that $x_1 \stackrel{a}{\Rightarrow} \nu_1$ iff there exists ν_2 such that $x_2 \stackrel{a}{\Rightarrow} \nu_2$, and when they do exist, there exists a real number δ in $[0, \epsilon']$ such that $\nu_1 \; \mathcal{L}(\mathcal{R}^{\epsilon'-\delta}, \delta) \; \nu_2$.

Lemma 4.3 *For all ϵ-unwinding families $\mathcal{R}^{\cdot}$, all ϵ' in $[0, \epsilon]$, all x_1 and x_2 in $S_\perp$ such that $x_1 \; \mathcal{R}^{\epsilon'} \; x_2$, all $\boldsymbol{i}$ in I^*, and all $\boldsymbol{e}$ in E^*, both*
$\Pr[\, \lfloor [\![\langle L, x_1 \rangle]\!](\boldsymbol{i}) \rfloor_E \sqsupseteq \boldsymbol{e} \,] \leq \exp(\epsilon') \Pr[\, \lfloor [\![\langle L, x_2 \rangle]\!](\boldsymbol{i}) \rfloor_E \sqsupseteq \boldsymbol{e} \,]$ *and*
$\Pr[\, \lfloor [\![\langle L, x_2 \rangle]\!](\boldsymbol{i}) \rfloor_E \sqsupseteq \boldsymbol{e} \,] \leq \exp(\epsilon') \Pr[\, \lfloor [\![\langle L, x_1 \rangle]\!](\boldsymbol{i}) \rfloor_E \sqsupseteq \boldsymbol{e} \,]$.

The above lemma shows that two states related by an ϵ-unwinding family, given the same input sequence, produce distributions that only deviate by a factor ϵ. Thus, to maintain ϵ-differential noninterference, we desire that a state s should upon receiving a single data point d transition to a state s' that can be put into an ϵ-unwinding family with s. We formalize this intuition with the next definition and confirm it with the following theorem.

Definition 4.4 [Covers] We say that an ϵ-unwinding family $\mathcal{R}^{\cdot}$ for a PLTS L *covers* a state s and data point d of L if $s \stackrel{d}{\Rightarrow} \nu$ implies that $\nu(\perp) = 0$ and for all $s' \in \mathsf{Supp}(\nu)$, $s \; \mathcal{R}^{\epsilon} \; s'$.

Theorem 4.5 *For an automaton $M = \langle L, s_0 \rangle$, if for all H-disabled states s reachable from s_0 and all data points d, there exists a ϵ-unwinding family that covers s and d, then $[\![M]\!]$ has ϵ-differential noninterference.*

Our related technical report [45] holds the proofs of Lemma 4.3 and Theorem 4.5. We prove Lemma 4.3 by induction over the structure of $\boldsymbol{a}$. The interesting cases arise when $\boldsymbol{a}$ is of the form $i{:}\boldsymbol{a}'$ for $i \in I$ or $o{:}\boldsymbol{a}'$ for $o \in O$, which require similar reasoning. Suppose that $\boldsymbol{a} = i{:}\boldsymbol{a}'$ and $s_1 \stackrel{i}{\Rightarrow} \nu_1$ for some $i \in I$. By the unwinding relation, we know that there exists a transition $s_2 \stackrel{i}{\Rightarrow} \nu_2$ such that ν_1 and ν_2 are in keeping with the privacy leakage bound imposed by the unwinding relation. Then for states $s_1' \in \mathsf{Supp}(\nu_1)$, and $s_2' \in \mathsf{Supp}(\nu_2)$, we apply the inductive hypothesis for $\boldsymbol{a}'$ to obtain the result.

To prove Theorem 4.5, we show for all $\boldsymbol{i}_1$, $\boldsymbol{i}_2$, and $\boldsymbol{e}$ where $\Delta(\boldsymbol{i}_1, \boldsymbol{i}_2) = 1$ that $\Pr[\lfloor [\![\langle L, s \rangle]\!](\boldsymbol{i}_1) \rfloor_E \sqsupseteq \boldsymbol{e}] \leq \exp(\epsilon) \Pr[\lfloor [\![\langle L, s \rangle]\!](\boldsymbol{i}_2) \rfloor_E \sqsupseteq \boldsymbol{e}]$. We use proof by induction over $\boldsymbol{i}_1$, $\boldsymbol{i}_2$, and $\boldsymbol{e}$. When we reach the point where $\boldsymbol{i}_1$ and $\boldsymbol{i}_2$ differ by a data point d, we apply Lemma 4.3 knowing that an ϵ-unwinding family exists for the current state s and d.

4.2 Example: Applying the Proof Technique

We now return to the system presented in Section 2.2 and modeled as a parametric automaton $M_{\mathsf{ex1}}(K)$ in Section 3.3. We will present an unwinding relation that proves that the automaton has differential noninterference.

Given that the system uses ϵ-differentially private functions t times, one might be surprised that we prove that it has $(2t{*}\epsilon)$-differential noninterference rather than $(t * \epsilon)$-differential noninterference. This extra leakage comes from dealing with the bounded memory of actual computers. In particular, each array in `dPts` is limited to a length of `maxPts`. The program keeps track of the current number of data points stored in each slot with the array `numPts`. If the current slot has reached `maxPts` data points, the program drops any incoming data points until `curSlot` advances. This dropping of data points introduces extra privacy leakage since a single data point can have two effects: (1) the data point is included in calculations affecting the probabilities of some outputs, and (2) the data point's presence can cause the system to drop a future data point and exclude it from calculations. Thus, the system has only $(2t * \epsilon)$-differential noninterference. In many scenarios, the possibility of running out of memory for storing data points is unrealistic. If the number of data points can never reach the memory bound, then under this assumption, one can show that system has $(t * \epsilon)$-differential noninterference. [3]

To prove that the automaton has $(2t * \epsilon)$-differential noninterference, we show that for any K, every state s and data point d of $L_{\mathsf{ex1}}(K)$ can be *covered* by a $(2t{*}\epsilon)$-unwinding family $\mathcal{R}^{\cdot}_{s,d}$ in the sense of Definition 4.4. Differential noninterference will follow from Theorem 4.5.

For the $(2t * \epsilon)$-unwinding family $\mathcal{R}^{\cdot}_{s,d}$, we construct for each j in $[0, t]$ the unwinding relation $\mathcal{R}^{2j*\epsilon}_{s,d}$. To construct these unwinding relations, we first introduce

[3] To avoid this extra privacy leakage, it may be tempting to use a linked list for each slot and keep track of how many total data points are stored in all the slots combined. Then, the program could drop data points only when all the memory is exhausted instead of just the current slot's allocation. However, this change would allow a single data point stored in one slot to affect which data points are dropped from other slots in the future. Thus, a single data point may have an unbounded effect on future computation preventing such a program from satisfying differential noninterference for any privacy bound.

some notation.

For a state $s = \langle pc, \boldsymbol{B}, \boldsymbol{n}, c, y, r, k\rangle$ and $d \in D$, $\mathsf{add}(s, c', d)$ adds d to the slot c' of the state s. Formally,

$$\mathsf{add}(s, c', d) = \langle pc, \boldsymbol{B}', \boldsymbol{n}, c, y, r, k\rangle$$

where $\boldsymbol{B}' = \boldsymbol{B}$ and $\boldsymbol{n}' = \boldsymbol{n}$ when $n_c = v$ and, otherwise, $B'_c = B_c \uplus \{d\}$, $n'_c = n_c + 1$, and for all $c' \neq c$, $B'_{c'} = B_{c'}$ and $n'_{c'} = n_{c'}$.

The function swap replaces one data point with another. Formally,

$$\mathsf{swap}(s, c', d, d') = \langle pc, \boldsymbol{B}', \boldsymbol{n}, c, y, r, k\rangle$$

where $B'_{c'} = B_{c'} - \{d'\} \uplus \{d\}$ and $B'_{c''} = B_{c''}$ for all $c'' \neq c'$.

For j such that $0 \leq j \leq t$, let S_1^j to be the set of all states s_1 such that s_1 is reachable from s using $t - j$ queries and any number of data points. Intuitively, this means that from s_1 one can pose j more queries until the privacy budget runs out on the data point that is input into the system in state s. We define the relations as follows:

- For $j > 0$, let $\mathcal{R}_{s,d}^{2j*\epsilon}$ to be such that for all $s_1 \in S_1^j$, $s_1 \; \mathcal{R}_{s,d}^{2j*\epsilon} \; \mathsf{add}(s_1, c, d)$ and for all d', $s_1 \; \mathcal{R}_{s,d}^{2j*\epsilon} \; \mathsf{swap}(s_1, c, d, d')$ where $s = \langle pc, \boldsymbol{B}, \boldsymbol{n}, c, y, r, k\rangle$. That is, $\mathcal{R}_{s,d}^{2j*\epsilon}$ relates a state to the states it could have become had it received d as input when the `curSlot` was c, the value `curSlot` had in state s.
- For $j = 0$, $\mathcal{R}_{s,d}^{2j*\epsilon}$ is as above for states with a PC of 16 and is equality for those with a PC of 08.

Lemma 4.6 *For all sets K of functions such that each function in K has ϵ-differential privacy, for all states s and for all data points d, $\mathcal{R}_{s,d}^{\cdot}$ is a $(2t * \epsilon)$-unwinding family for the automaton $M_{\mathsf{ex1}}(K)$.*

Our related technical report [45] holds the proof. The proof uses a case analysis over the different types of actions a that might be received by two related states. The most interesting case is when a is a query and $j = 1$. In this case, $s_1 \; \mathcal{R}_{s,d}^{\epsilon'} \; s_2$ implies that s_1 is in S_1^{t-1} with s_1 and s_2 reached in $t - 1$ queries. For a $2t * \epsilon$ privacy leakage bound, this corresponds to the last time d may be used in answering a query. This requirement is met since for s_1 and s_2 to be reached with $t - 1$ queries, by the construction of $M_{\mathsf{ex1}}(K)$, `curSlot` in both states must be $t - 1$ slots away from the slot that holds d. Thus, after answering the next query the slot `curSlot`, whose value is always mod t, will point to the slot that holds d and that slot will be rewritten removing d.

Since $\mathcal{R}_{s,d}^{2j*\epsilon}$ covers s and d for all states s and data points d of the automaton $M_{\mathsf{ex1}}(K)$, Lemma 4.6 and Theorem 4.5 implies that the automaton has $(2t * \epsilon)$-differential noninterference.

Theorem 4.7 *For all set of functions K such that each function in K has ϵ-differential privacy, $M_{\mathsf{ex1}}(K)$ has $(2t * \epsilon)$-differential noninterference.*

As COUNT and SUM are ϵ-differentially private functions, this implies that $M_{\mathsf{ex1}}(\{\text{COUNT}, \text{SUM}\})$ has $(2t * \epsilon)$-differential noninterference.

5 Related Work

Formal Verification of Differential Privacy. The most closely related work to ours is a programming language with a linear type system for proving that well-typed programs in the language have differential privacy [38]. Later work applies their type system to detecting network attacks in a private manner [37]. The usual trade-offs between a program analysis technique designed to work over standard programming languages and a custom type system for a specialized language apply: the type system makes explicit in the source code why the program has differential privacy and type checking scales well, but the programmer must use a special-purpose programming language and annotate the code as the type system requires. Additionally, their programming language lacks I/O commands for creating interactive systems whereas our proof technique is for automata modeling interactive systems.

Other Differential Privacy Definitions. The definition of differential privacy may be seen as largely a simplification of the previously defined notion of ϵ-*indistinguishability* [17], which explicitly models interaction between a private system and the data examiner as in our definition of differential noninterference. Our definition, however, is cast in the framework of probabilistic automata rather than Turing machines. This supports having structured models that are capable of highlighting issues arising from the bounded memory of actual computers. Furthermore, we deal with non-termination using prefixes allowing us to leverage previous work on formal methods for automata (e.g., [25]).

Differential privacy is a very active research field giving rise to new definitions and techniques at a fast pace [16,18]. For example, *pan-privacy* is a notion of differential privacy that gives differential privacy against adversaries that can observe the internal state of a system, in addition to outputs [32]. *Computational differential privacy* gives certain differential privacy guarantees against computationally bounded adversaries. Our definition of differential noninterference and the formal proof technique was developed from the definition of Dwork [13]. We think that our choice of probabilistic automata as a model would prove useful in extending the work of this paper to these new definitions as well. For example, algorithms such as stream-processing algorithms that have been subject to research from pan-privacy point of view can be naturally modeled using probabilistic automata. Similarly, probabilistic automata-based models have successfully been used in the formal analysis of cryptographic protocols against computationally bounded adversaries [42,3,8].

Information-Flow Properties. Differential noninterference has some similarities with information flow properties such as noninterference [21]. The literature contains several works on the use of transition systems, observational equivalences, and various notions of bisimulation relations to define information flow

properties. To name a few, Focardi and Gorrieri have developed a classification of noninterference-like properties in the unifying framework of a process algebra in a non-probabilistic setting [19]. Sabelfeld and Sands [40], and Smith [43] have used probabilistic bisimulation in defining probabilistic noninterference for multi-threaded programs, which they enforce using type systems. Probabilistic noninterference is regarded by many to be too strong in practice since it 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) [23,24]. As noninterference is often too strong of a requirement, weaker probabilistic versions have been proposed that allow for some information leakage [36,2]. Di Pierro, Hankin, and Wiklicky introduced *approximate noninterference* [36], and Backes and Pfitzmann introduced *computational probabilistic noninterference* [2], both of which allow for some information leakage. However, unlike differential noninterference, they do not allow the system behavior to diverge as the difference between the high-level inputs (data points) increases. This divergence, which is allowed by our differential noninterference definition, is needed to release meaningful statistics and gain utility from the data set as discussed in detail in Section 1.

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 [11]. 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 [27], and Chen and Malacaria to multi-threaded programs [10]. McCamant and Ernst [28], and Newsome and Song [33] provide dynamic analyses for quantitative information flow using the mutual information formalization. There is also recent work on efficient computation of information leakage in the information theoretic-sense using a probabilistic automaton model [1]. 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 instead propose a formulation using the beliefs of the adversary [12]. 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.

Proof Techniques for Transition Systems. Simulation and bisimulation provide a systematic proof technique for showing implementation and equivalence relationships between two automata [31,26,41] and are related to unwinding (see e.g., [7]). Most similar to our unwinding technique, Segala and Turrini have studied approximate simulation relations in the context of cryptographic protocols [42]. Their work differs from ours by using asymptotic approximations and only executions of polynomial length in terms of a security parameter. Their work allows certain transitions of the protocol to not have a matching transition in the specifi-

cation. This models the capability of the adversary to compromise correctness. A protocol is deemed correct if the leakage 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 an approximately matching transition, uses an exact error bound, and considers executions of any length. However, the probabilities of those transitions are only within some exponential multiplicative factor of one another. Thus, neither approach subsumes the other. Furthermore, our relations are over states whereas theirs is over prefixes of executions.

6 Future Work

The results of this paper represent progress towards developing a basis for the formal verification of differential privacy for systems, but leave open several interesting directions that we plan to explore in future work. While our related technical report [45] provides an algorithm for mechanically checking a restricted class of relations from the proof technique, we hope, in addition, to create a decision procedure for our proof technique by extending prior work on decision procedures for probabilistic bisimulations [5,4,35,9] to make them produce a family of relations rather than a single one. We also plan to extend the theory to model and reason about higher level systems, such as computer systems of hospitals and other distributed systems [39] that allow interactions of the system with data providers and with data analysts, while protecting the privacy of the data stored and manipulated by the system. For example, AIRAVAT allows computations over data distributed in a cloud, and combines mandatory access control with differential privacy where differential privacy is used to facilitate declassification governed by the privacy error bound set by a data provider. Our techniques can currently apply to the verification of differential privacy property of the AIRAVAT system using a whole-system model. We are interested in exploring the computational model of AIRAVAT further to understand the interplay between the fine-grained access control mechanisms and the differential privacy mechanisms in stating the end-to-end information-flow guarantee of AIRAVAT.

References

[1] Andres, M. E., C. Palamidessi, P. van Rossum and G. Smith, *Computing the leakage of information-hiding systems*, in: *Proceedings of Sixteenth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS **6015** (2010), pp. 373–389.

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

[3] Backes, M., B. Pfitzmann and M. Waidner, *The reactive simulatability framework for asynchronous systems*, Information and Computation (2007), preprint on IACR ePrint 2004/082.

[4] Baier, C., B. Engelen and M. Majster-Cederbaum, *Deciding bisimilarity and similarity for probabilistic processes*, Journal of Computer and System Sciences **60** (2000), pp. 187–231.

[5] Baier, C., H. Hermanns and J.-P. Katoen, *Probabilistic weak simulation is decidable in polynomial time*, Information Processing Letters **89** (2004), pp. 123–152.

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

[7] Bossi, A., R. Focardi, C. Piazza and S. Rossi, *Bisimulation and unwinding for verifying possibilistic security properties*, in: *VMCAI 2003: Proceedings of the 4th International Conference on Verification, Model Checking, and Abstract Interpretation* (2003), pp. 223–237.

[8] Canetti, R., L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira and R. Segala, *Time-bounded task-pioas: A framework for analyzing security protocols*, Journal of Discrete Event Dynamic Systems **18** (2008), pp. 111–159, short version appeared In 20th Symposium on Distributed Computing (DISC), 2006.

[9] Cattani, S. and R. Segala, *Decision algorithms for probabilistic bisimulation*, in: L. Brim, P. Jancar, M. Kretínský and A. Kucera, editors, *CONCUR '02: Proceedings of the 13th International Conference on Concurrency Theory*, LNCS **2421** (2002), pp. 371–385.
URL citeseer.ist.psu.edu/cattani02decision.html

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

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

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

[13] Dwork, C., *Differential privacy*, in: *33rd International Colloquium on Automata, Languages and Programming (ICALP 2006)*, 2006, pp. 1–12, part 2.

[14] Dwork, C., *Differential privacy: a survey of results*, in: *Proceedings of the 5th international conference on Theory and applications of models of computation*, TAMC'08 (2008), pp. 1–19.

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

[16] Dwork, C., *Differential privacy in new settings*, in: *Proceedings of Symposium on Discrete Algorithms (SODA)* (2010).

[17] Dwork, C., F. Mcsherry, K. Nissim and A. Smith, *Calibrating noise to sensitivity in private data analysis*, in: *Theory of Cryptography Conference*, Lecture Notes in Computer Science **3876** (2006), pp. 265–284.

[18] Dwork, C., M. Naor, T. Pitassi and G. N. Rothblum, *Differential privacy under continual observation*, in: *In Proceedings of the 42nd ACM Syposium on the Theory of Computing (STOC)*, 2010.

[19] Focardi, R. and R. Gorrieri, *Classification of security properties (Part I: Information flow)* (2001).

[20] Ghosh, A., T. Roughgarden and M. Sundararajan, *Universally utility-maximizing privacy mechanisms*, in: *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing* (2009), pp. 351–360.

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

[22] Goguen, J. A. and J. Meseguer, *Unwinding and inference control*, in: *Proc. of IEEE Symp. on Security and Privacy* (1984), pp. 75–86.

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

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

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

[26] Lynch, N. and F. Vaandrager, *Forward and backward simulations Part I: Untimed systems*, Inf. Comput. **121** (1995), pp. 214–233.

[27] 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* (2007), pp. 225–235.

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

[29] 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* (2009).

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

[31] Milner, R., "Communication and Concurrency," Prentice Hall, 1989.

[32] Mironov, I., O. Pandey, O. Reingold and S. Vadhan, *Computational differential privacy*, in: *Advances in Cryptology – CRYPTO 2009*, 2009.

[33] Newsome, J. and D. Song, *Influence: A quantitative approach for data integrity*, Technical Report CMU-CyLab-08-005, CyLab, Carnegie Mellon University (2008).

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

[35] Philippou, A., I. Lee and O. Sokolsky, *Weak bisimulation for probabilistic systems*, in: *CONCUR '00: Proceedings of the 11th International Conference on Concurrency Theory*, Lecture Notes in Computer Science **1877** (2000), pp. 334–349.

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

[37] Reed, J., A. J. Aviv, D. Wagner, A. Haeberlen, B. C. Pierce and J. M. Smith, *Differential privacy for collaborative security*, in: *European Workshop on System Security (EUROSEC)*, 2010.

[38] Reed, J. and B. C. Pierce, *Distance makes the types grow stronger: A calculus for differential privacy*, in: *ACM SIGPLAN International Conference on Functional Programming (ICFP)*, 2010.

[39] Roy, I., H. E. Ramadan, S. T. Setty, A. Kilzer, V. Shmatikov and E. Witchel, *Airavat: Security and privacy for MapReduce*, in: *Proceedings of the 7th Usenix Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[40] Sabelfeld, A. and D. Sands, *Probabilistic non-interefence for multi-threaded programs*, in: *Proceedings of the 13th IEEE Computer Security Foundations Workshop* (July 2000).

[41] Segala, R. and N. Lynch, *Probabilistic simulations for probabilistic processes*, Nordic Journal of Computing **2** (1995).

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

[43] Smith, G., *Probabilistic noninterference through weak probabilistic bisimulation*, in: *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, Pacific Grove, California, 2003, pp. 3–13.

[44] Tschantz, M. C., A. Datta and D. Kaynar, *Differential privacy for probabilistic systems*, Technical Report CMU-CyLab-09-008, CyLab, Carnegie Mellon University (2009), http://www.cylab.cmu.edu/files/pdfs/tech_reports/CMUCylab09008.pdf.

[45] Tschantz, M. C., D. Kaynar and A. Datta, *Formal verification of differential privacy for interactive systems*, CoRR **abs/1101.2819** (2011), http://arxiv.org/abs/1101.2819.