Probabilistic Relational Reasoning for Differential Privacy

Gilles Barthe   Boris Köpf   Federico Olmedo   Santiago Zanella Béguelin
IMDEA Software Institute, Madrid, Spain
{gilles.barthe,boris.koepf,federico.olmedo,santiago.zanella}@imdea.org

Abstract

Differential privacy is a notion of confidentiality that protects the privacy of individuals while allowing useful computations on their private data. Deriving differential privacy guarantees for real programs is a difficult and error-prone task that calls for principled approaches and tool support. Approaches based on linear types and static analysis have recently emerged; however, an increasing number of programs achieve privacy using techniques that cannot be analyzed by these approaches. Examples include programs that aim for weaker, approximate differential privacy guarantees, programs that use the Exponential mechanism, and randomized programs that achieve differential privacy without using any standard mechanism. Providing support for reasoning about the privacy of such programs has been an open problem.

We report on CertiPriv, a machine-checked framework for reasoning about differential privacy built on top of the Coq proof assistant. The central component of CertiPriv is a quantitative extension of a probabilistic relational Hoare logic that enables one to derive differential privacy guarantees for programs from first principles. We demonstrate the expressiveness of CertiPriv using a number of examples whose formal analysis is out of the reach of previous techniques. In particular, we provide the first machine-checked proofs of correctness of the Laplacian and Exponential mechanisms and of the privacy of randomized and streaming algorithms from the recent literature.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory; F.3.1 [*Logics and Meanings of Programs*]: Specifying and Verifying and Reasoning about Programs; F.3.2 [*Logics and Meanings of Programs*]: Semantics of Programming Languages—Operational semantics, Denotational semantics, Program analysis.

General Terms Languages, Security, Theory, Verification

Keywords Coq proof assistant, differential privacy, relational Hoare logic

1. Introduction

When dealing with private data one is faced with conflicting requirements: on the one hand, it is fundamental to protect the privacy of individuals; on the other hand, the desire is to maximize the utility of the data by mining and releasing partial or aggregate information, e.g. for medical statistics, market research, or targeted advertising. Differential privacy [17] is a quantitative notion of privacy that achieves an attractive trade-off between these two conflicting requirements: it provides strong confidentiality guarantees, yet it is permissive enough to allow for useful computations on private data. The key advantages of differential privacy over alternative definitions of privacy are its good behavior under composition and its weak assumptions about the prior knowledge of adversaries. For a discussion of the guarantees provided by differential privacy and their limitations, see [21, 22].

As the theoretical foundations of differential privacy become well-understood, there is momentum to prove privacy guarantees for real systems. Several authors have recently proposed methods for reasoning about differential privacy on the basis of different languages and models of computation, e.g. SQL-like languages [24], higher-order functional languages [29], imperative languages [9], the MapReduce framework [30], and I/O automata [35]. The unifying basis of these approaches are two key results: (i) the observation that one can achieve privacy by perturbing the output of a deterministic program by a suitable amount of noise [17] and (ii) theorems that establish privacy bounds for sequential and parallel composition of differentially private programs [24]. In combination, both results form the basis for creating and analyzing programs by composing differentially private building blocks.

While approaches relying on composing building blocks apply to an interesting range of examples, they fall short of covering the expanding frontiers of differentially private mechanisms and algorithms. Examples that cannot be handled by previous techniques include mechanisms that aim for weaker guarantees, such as approximate differential privacy [16], or randomized algorithms that achieve differential privacy without using any standard mechanism [18]. Dealing with such examples requires fine-grained reasoning about the complex mathematical and probabilistic computations that programs perform on private input data. Such reasoning is particularly intricate and error-prone, and calls for principled approaches and tool support.

In this paper we revisit the foundations of differential privacy and provide a framework for fine-grained reasoning about an expressive class of confidentiality policies, including (approximate) differential privacy and probabilistic non-interference. Our framework, coined CertiPriv, is built on top of CertiCrypt [3], a machine-checked framework to verify cryptographic proofs in the Coq proof assistant [34]. CertiPriv goes beyond the state-of-the-art in three fundamental aspects. First, CertiPriv takes a foundational approach that allows reasoning directly about the outcome of probabilistic computations. This is key to its flexibility: rather than being limited to a fixed set of building blocks, one can define and use arbitrary blocks. Second, CertiPriv allows to construct proofs from first principles. This is key to its precision: proofs in CertiPriv can rely on sophisticated machinery, without any limitation other than being elaborated from first principles. Third, CertiPriv inherits the generality of the Coq proof assistant and allows modeling and reasoning about arbitrary domains and datatypes. This is key to its expressiveness: instead of being confined to a fixed set of datatypes, CertiPriv can be extended on demand (e.g. with types and operators for graphs). Accessorily, CertiPriv requires that all intermediate reasoning steps are justified formally, so that proofs can be verified independently and automatically by the Coq type checker.

In order to illustrate the applicability of CertiPriv, we present the first machine-checked proofs of three representative examples: (i) we prove the correctness of the Laplacian and Exponential mechanisms, (ii) we prove the privacy of a randomized approximation algorithm for the Minimum Vertex Cover problem [18], and (iii) we prove the privacy of randomized algorithms for continual release of aggregate statistics of data streams [8]. Taken together, these examples demonstrate the generality and versatility of our approach.

The starting point of our technical development is the observation that differential privacy can be construed as a quantitative 2-property [11, 33]. Informally, a probabilistic computation c is (ϵ, δ)-differentially private iff, given two initial memories m and m' that are sufficiently close, the output distributions generated by c are related up to a multiplicative factor $\exp(\epsilon)$ and an additive term δ. More formally, a computation c satisfies (ϵ, δ)-differential privacy with respect to a relation Ψ on memories iff for every pair of memories m, m' related by Ψ and for every event E:

$$\Pr\left[c, m : E\right] \leq \exp(\epsilon) \Pr\left[c, m' : E\right] + \delta$$

where $\Pr\left[c, m : E\right]$ denotes the probability of event E in the distribution obtained by running c on initial memory m. This formulation of differential privacy is slightly more general than the standard definition; however, the latter is recovered by letting the precondition Ψ capture adjacency of memories, i.e. letting Ψ relate memories at distance at most 1 for some adequate notion of distance.

Our definition of differential privacy has two natural readings. The first reading is as an information flow property. Indeed, if Ψ is an equivalence relation and $\epsilon = \delta = 0$, the definition states that the output distributions obtained by executing c in two related memories m and m' coincide, entailing that an adversary who can only observe the final distributions cannot distinguish between the two executions. The second reading of the definition is as a continuity property: in case Ψ models adjacency between initial memories, the definition states that c is a continuous mapping between metric spaces, with the understanding that the universally quantified inequality above provides a measure of closeness of the two output distributions. In this paper, we leverage on both readings to provide a fresh foundation for reasoning about differentially private computations.

As a first step in our formalization, we introduce the notion of α-distance. α-distance generalizes statistical distance with a skew parameter α and enables us to cast (ϵ, δ)-differential privacy as a continuity property. In particular, we show that a computation c is (ϵ, δ)-differentially private w.r.t. a pre-condition Ψ iff δ is an upper bound of the $\exp(\epsilon)$-distance between the output distributions obtained by running c on two memories m and m' satisfying Ψ.

As a second step, we define an approximate probabilistic Relational Hoare Logic (apRHL), following Benton's seminal use of relational logics to reason about information flow [7]. Judgments in apRHL have the form

$$c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi$$

and capture that δ is an upper bound on the α-distance of the probability distributions generated by two probabilistic programs c_1 and c_2, modulo relational pre- and post-conditions Ψ and Φ on program states. For the special case where Φ is the identity on states, $c_1 = c_2 = c$, and $\alpha = \exp(\epsilon)$, the above judgment entails that the output distributions obtained by executing c starting from two initial memories related by Ψ are at α-distance at most δ, and hence that c is (ϵ, δ)-differentially private w.r.t. Ψ.

As further detailed in Section 5.2, this intuitive understanding of apRHL judgments extends to the important case where Φ is an equivalence relation; such judgments generalize simultaneously differential privacy and information flow and can be used to model confidentiality for a large class of adversaries, under the view that the equivalence relation captures their observational capabilities. For the general case, the interpretation of apRHL judgments is based on the novel notion of (α, δ)-lifting of relations on states to relations on distributions. The definition crisply generalizes existing notions from probabilistic process algebra [12, 20, 32] and enjoys good closure properties from which we derive the soundness of the apRHL logic.

Summary of contributions Our contributions are twofold. On the theoretical side, we lay the foundations for reasoning formally about an important and general class of approximate relational properties of probabilistic programs. Specifically, we introduce the notions of α-distance and (α, δ)-lifting, and an approximate probabilistic relational Hoare logic. On the practical side, we demonstrate the applicability of our approach by providing the first machine-checked proofs of differential privacy properties of fundamental mechanisms and complex approximation algorithms from the recent literature.

Organization of the paper The remainder of this paper is structured as follows. In Section 2 we illustrate the application of our approach to an example algorithm; Section 3 introduces the representation of distributions and basic definitions used in the remainder. Section 4 presents the semantic foundations of apRHL, while Section 5 presents the core proof rules of the logic. Section 6 reports on case studies. We survey prior art and conclude in Sections 7 and 8. The Coq development containing machine-checked proofs of the results and examples presented here, and an extended version of this paper with pencil-and-paper proofs of the key results can be obtained from

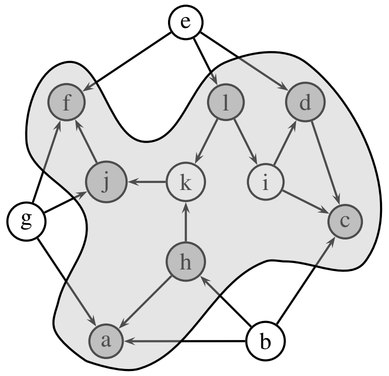
```
http://certicrypt.gforge.inria.fr/certipriv/
```

2. Illustrative Example

In this section we illustrate the applicability of our results by analyzing a differentially private approximation algorithm for the Minimum (Unweighted) Vertex Cover problem [18].

A *vertex cover* of an undirected graph $G = (V, E)$ is a set of vertices $S \subseteq V$ such that for any edge $(v, w) \in E$, either $v \in S$ or $w \in S$. The Minimum Vertex Cover problem is the problem of finding a vertex cover S of minimal size. In the privacy-preserving version of the problem the goal is to output a good approximation of a minimum cover while concealing the presence or absence of edges in the graph. Contrary to other optimization algorithms where the private data only determines the objective function (i.e. the size of a minimum cover), in the case of the Minimum Vertex Cover problem the edges in the graph determine the feasible solutions. This means that no privacy-preserving algorithm can explicitly output a vertex cover of size less than $n-1$ for a graph with n vertices, for otherwise any pair of vertices absent from the output reveals the absence of an edge connecting them. To overcome this limitation, the algorithm that we analyze outputs an implicit representation of a cover as a permutation of the vertices in the graph. This output permutation determines an orientation of the edges in the graph by considering each edge as pointing towards the endpoint appearing last in the permutation. A vertex cover can then be recovered (presumably in a privacy-preserving distributed manner) by taking for each edge the vertex it points to (Fig. 1).

The algorithm that we study, shown in Fig. 2, is based on a randomized, albeit not privacy-preserving, approximation algorithm from [27] that achieves a constant approximation factor of 2. (It is conjectured that no efficient approximation algorithm for the Minimum Vertex Cover problem can achieve a constant approximation factor better than 2.) The idea behind this algorithm is to iteratively pick a random uncovered edge and add one of its endpoints to the cover set, both the edge and the endpoint being chosen with uniform probability. Equivalently, this iterative process can be seen



$\pi = [b, g, e, h, l, k, j, i, f, d, c, a]$

Figure 1. A minimum vertex cover (vertices in gray) and the cover given by a permutation π of the vertices in the graph (vertices inside the shaded area). The orientation of the edges is determined by π.

as selecting a vertex at random with probability proportional to its uncovered degree. The privacy-preserving algorithm in Fig. 2 is obtained from this base algorithm by perturbing the distribution according to which vertices are sampled by a carefully calibrated weight factor that grows as more vertices are appended to the output permutation. In the algorithm in the figure, at each iteration the instruction $v \xleftarrow{\$} \mathsf{choose}(V, E, \epsilon, n, i)$ chooses a vertex v from V with probability proportional to $d_E(v) + w_i$, where $d_E(v)$ denotes the degree of v in E and

$$w_i = \frac{4}{\epsilon}\sqrt{\frac{n}{n-i}}$$

Put otherwise, the expression $\mathsf{choose}(V, E, \epsilon, n, i)$ denotes the discrete distribution over V whose density function at x is

$$\frac{d_E(x) + w_i}{\sum_{y \in V} d_E(y) + w_i}$$

Consider two graphs $G_1 = (V, E)$ and $G_2 = (V, E \cup \{(t, u)\})$ with the same set of vertices but differing in exactly one edge. To prove that the above algorithm is ϵ-differentially private it is sufficient to show that the probability of obtaining a permutation π of the vertices in the graph when the input is G_1 differs at most by a multiplicative factor $\exp(\epsilon)$ from the probability of obtaining π when the input is G_2. We show this using the approximate relational Hoare logic that we present in Section 5. We highlight here the key steps in the proof; a more detailed account appears in Section 6.3.

To establish the ϵ-differential privacy of algorithm VERTEXCOVER it suffices to prove the validity of the following judgment:

$$\models \textsc{VertexCover}(V, E, \epsilon) \sim_{e^\epsilon, 0} \textsc{VertexCover}(V, E, \epsilon) : \Psi \Rightarrow \Phi$$

where

$$\Psi \overset{\text{def}}{=} V\langle 1\rangle = V\langle 2\rangle \wedge E\langle 2\rangle = E\langle 1\rangle \cup \{(t, u)\}$$
$$\Phi \overset{\text{def}}{=} \pi\langle 1\rangle = \pi\langle 2\rangle$$

Assertions appearing in apRHL judgments, like Ψ and Φ above, are binary relations on program memories. We usually define assertions using predicate logic formulas involving program expressions. When defining an assertion $m_1\ \Phi\ m_2$, we denote by $e\langle 1\rangle$ (resp. $e\langle 2\rangle$) the value that the expression e takes in memory m_1 (resp. m_2). For example, the post-condition Φ above denotes the relation $\{(m_1, m_2) : m_1(\pi) = m_2(\pi)\}$.

To prove the judgment above, we show privacy bounds for each iteration of the loop in the algorithm. Proving a bound for the i-th iteration boils down to proving a bound for the ratio between the probability of choosing a particular vertex in the left-hand side program and the right-hand side program, and its reciprocal. We distinguish three different cases, and use the fact that for a graph (V, E), $\sum_{y \in V} d_E(y) = 2|E|$ and the inequality $1 + x \leq \exp(x)$ to derive upper bounds in each case:

```
function VERTEXCOVER($V, E, \epsilon$)
  $n \leftarrow |V|$; $\pi \leftarrow$ nil; $i \leftarrow 0$;
  while $i < n$ do
    $v \xleftarrow{\$}$ choose($V, E, \epsilon, n, i$);
    $\pi \leftarrow v :: \pi$;
    $V \leftarrow V \setminus \{v\}$; $E \leftarrow E \setminus (\{v\} \times V)$;
    $i \leftarrow i + 1$
  end
```

Figure 2. A differentially private approximation algorithm for the Minimum Unweighted Vertex Cover problem

(a) the chosen vertex is not one of t, u and neither t nor u are in π.

$$\begin{aligned}\frac{\Pr[v\langle 1\rangle = x]}{\Pr[v\langle 2\rangle = x]} &= \frac{(d_{E\langle 1\rangle}(x) + w_i)\sum_{y\in V}(d_{E\langle 2\rangle}(y) + w_i)}{(d_{E\langle 2\rangle}(x) + w_i)\sum_{y\in V}(d_{E\langle 1\rangle}(y) + w_i)} \\ &= \frac{(d_{E\langle 1\rangle}(x) + w_i)(2|E\langle 1\rangle| + (n-i)w_i + 2)}{(d_{E\langle 1\rangle}(x) + w_i)(2|E\langle 1\rangle| + (n-i)w_i)} \\ &\leq 1 + \frac{2}{(n-i)w_i} \leq \exp\left(\frac{2}{(n-i)w_i}\right)\end{aligned}$$

$$\frac{\Pr[v\langle 2\rangle = x]}{\Pr[v\langle 1\rangle = x]} \leq 1$$

(b) the vertex v chosen in the iteration is one of t, u. We analyze the case where $v = t$, the other case is similar.

$$\frac{\Pr[v\langle 1\rangle = t]}{\Pr[v\langle 2\rangle = t]} \leq 1$$

$$\begin{aligned}\frac{\Pr[v\langle 2\rangle = t]}{\Pr[v\langle 1\rangle = t]} &= \frac{(w_i + d_{E\langle 1\rangle}(t) + 1)(2|E\langle 1\rangle| + (n-i)w_i)}{(w_i + d_{E\langle 1\rangle}(t))(2|E\langle 1\rangle| + (n-i)w_i + 2)} \\ &\leq 1 + w_i^{-1} \leq 1 + w_0^{-1} \leq \exp(\epsilon/4)\end{aligned}$$

(c) either t or u is already in π, in which case both executions are observationally equivalent and do not add to the privacy bound.

$$\frac{\Pr[v\langle 1\rangle = x]}{\Pr[v\langle 2\rangle = x]} = \frac{\Pr[v\langle 2\rangle = x]}{\Pr[v\langle 1\rangle = x]} = 1$$

Case (a) can occur at most $(n-2)$ times, while case (b) occurs exactly once. Thus, multiplying the bounds over all n iterations,

$$\begin{aligned}\frac{\Pr[\textsc{VertexCover}(G_1, \epsilon) : \pi = \vec{v}]}{\Pr[\textsc{VertexCover}(G_2, \epsilon) : \pi = \vec{v}]} &\leq \exp\left(\sum_{i=0}^{n-3}\frac{2}{(n-i)w_i}\right) \\ &\leq \exp(\epsilon)\end{aligned}$$

$$\frac{\Pr[\textsc{VertexCover}(G_2, \epsilon) : \pi = \vec{v}]}{\Pr[\textsc{VertexCover}(G_1, \epsilon) : \pi = \vec{v}]} \leq \exp(\epsilon/4) \leq \exp(\epsilon)$$

The above informal reasoning is captured by a proof rule for loops parameterized by an invariant and a stable property of the product state of both executions (i.e. a relation that once established remains true). We use the following invariant (note that if pre-condition Ψ above holds, the invariant is established by the initialization code appearing before the loop):

$$\begin{aligned}&(t \in \pi\langle 1\rangle \vee u \in \pi\langle 1\rangle \implies E\langle 1\rangle = E\langle 2\rangle) \wedge \\ &(t \notin \pi\langle 1\rangle \wedge u \notin \pi\langle 1\rangle \implies E\langle 2\rangle = E\langle 1\rangle \cup \{(t, u)\}) \wedge \\ &V\langle 1\rangle = V\langle 2\rangle \wedge \pi\langle 1\rangle = \pi\langle 2\rangle\end{aligned}$$

and the following stable property:

$$t \in \pi\langle 1\rangle \vee u \in \pi\langle 1\rangle$$

The application of this proof rule requires to prove three judgments as premises, corresponding to each one of the cases detailed above; we detail them in Section 6.3.

3. Preliminaries

3.1 Probabilities and Reals

In the course of our Coq formalization, we have found it convenient to reason about probabilities using the axiomatization of the unit interval $[0,1]$ provided by the ALEA library of Audebaud and Paulin [1]. Their formalization supports as primitive operations addition, inversion, multiplication, and division, and proves that the unit interval $[0,1]$ can be given the structure of a ω-cpo by taking as order the usual $\leq$ relation and by defining an operator $\sup$ that computes the least upper bound of monotonic $[0,1]$-valued sequences.

In order to manage the interplay between the formalizations of the unit interval and of the reals, we have axiomatized an embedding/retraction pair between them and built an extensive library of results about the relationship between arithmetic operations in the two types, e.g.:

Addition: $x +_{[0,1]} y = \min_{\mathbb{R}}(x +_{\mathbb{R}} y, 1)$;

Inversion: $-_{[0,1]}x = 1 -_{\mathbb{R}} x$;

Multiplication: $x \times_{[0,1]} y = x \times_{\mathbb{R}} y$;

Division: if $y \neq 0$, then $x/_{[0,1]}y = \min_{\mathbb{R}}(x/_{\mathbb{R}}y, 1)$.

3.2 Distributions

We view a distribution μ over a set A as a function that maps a unit-valued random variable (a function in $A \to [0,1]$) to its expected value [1, 28]: when applied to an event $E \subseteq A$ represented by its characteristic function $\mathbb{1}_E \colon A \to [0,1]$, $\mu(\mathbb{1}_E)$ corresponds to the probability of E. When applied to an arbitrary function $f : A \to [0,1]$, $\mu(f)$ gives the expectation of f w.r.t. μ. For discrete distributions μ, $\mu(\mathbb{1}_a)$ corresponds to the probability density of μ at a, and we denote it using the shorthand $\mu(a)$. The connection between density and expectation is given by the following equation.

$$\mu(f) = \sum_{a\in A} \mu(a)\, f(a)$$

Formally, a distribution over A is a function μ of type

$$(A \to [0,1]) \to [0,1]$$

together with proofs of the (universally quantified) properties:

Monotonicity: $f \leq g \implies \mu\, f \leq \mu\, g$;

Compatibility with inverse: $\mu\,(\mathbb{1} - f) \leq 1 - \mu\, f$, where $\mathbb{1}$ is the constant function 1;

Additive linearity: $f \leq \mathbb{1} - g \implies \mu\,(f+g) = \mu\, f + \mu\, g$;

Multiplicative linearity: $\mu\,(k \times f) = k \times \mu\, f$;

Continuity: if $F : \mathbb{N} \to (A \to [0,1])$ is monotonic, then $\mu\,(\sup\, F) \leq \sup\,(\mu \circ F)$

Note that we do not require that $\mu\ \mathbb{1} = 1$, and thus, strictly speaking, our definition corresponds to sub-probability distributions. This provides an elegant means of giving semantics to runtime assertions and programs that do not terminate with probability one. We let $\mathcal{D}(A)$ denote the set of distributions over A and μ_0 denote the null distribution.

Distributions can be given the structure of a monad; this monadic view eliminates the need of cluttered definitions and proofs involving summations, and allows to give a continuation-passing style semantics to probabilistic programs. Formally, we define the unit and bind operators as follows:

$$\begin{array}{l}
\mathsf{unit}\ :\ A \to \mathcal{D}(A) \overset{\text{def}}{=} \lambda x.\, \lambda f.\, f\ x \\
\mathsf{bind}\ :\ \mathcal{D}(A) \to (A \to \mathcal{D}(B)) \to \mathcal{D}(B) \\
\qquad \overset{\text{def}}{=} \lambda \mu.\, \lambda M.\, \lambda f.\, \mu\ (\lambda\, x.\, M\ x\ f)
\end{array}$$

The unit operator maps $x \in A$ to the Dirac distribution that assigns probability 1 to x and 0 to all other elements of A, while bind takes a distribution on A and a conditional distribution on B given A, and returns the corresponding marginal distribution on B.

In the remainder we use the following operations and relations:

$$\begin{array}{rcl}
\mathsf{range}\ P\ \mu & \overset{\text{def}}{=} & \forall f.\ (\forall a.\ P\ a \implies f\ a = 0) \implies \mu\ f = 0 \\
\pi_1(\mu) & \overset{\text{def}}{=} & \mathsf{bind}\ \mu\ (\lambda(x,y).\ \mathsf{unit}\ x) \\
\pi_2(\mu) & \overset{\text{def}}{=} & \mathsf{bind}\ \mu\ (\lambda(x,y).\ \mathsf{unit}\ y) \\
\mu \leq \mu' & \overset{\text{def}}{=} & \forall f.\ \mu\ f \leq \mu'\ f
\end{array}$$

The formula $\mathsf{range}\ P\ \mu$ states that elements of A with a non-null probability w.r.t. μ satisfy predicate P. For a distribution μ over a product type $A \times B$, $\pi_1(\mu)$ (resp. $\pi_2(\mu)$) defines its projection on the first (resp. second) component. Finally, $\leq$ defines a natural order on $\mathcal{D}(A)$.

4. First Principles

4.1 Skewed Distance of Distributions

In this section we define the notion of α-distance, a parameterized distance between distributions. We show how this notion can be used to express ϵ-differential privacy, (ϵ,δ)-differential privacy, and statistical distance.

We begin by augmenting the standard distance between two real numbers a and b (defined as $|a-b| = \max\{a-b, b-a\}$) with a skew parameter $\alpha \geq 1$. Namely, we define the α-distance $\Delta_\alpha(a,b)$ between a and b by

$$\Delta_\alpha(a,b) \overset{\text{def}}{=} \max\{a - \alpha b, b - \alpha a, 0\}$$

Note that Δ_α is non-negative by definition and that Δ_1 coincides with the standard distance between reals. We extend Δ_α to a distance between distributions as follows.

Definition 1 (α-distance)**.** *The α-distance $\Delta_\alpha(\mu_1,\mu_2)$ between two distributions μ_1 and μ_2 in $\mathcal{D}(A)$ is defined as:*

$$\Delta_\alpha(\mu_1,\mu_2) \overset{\text{def}}{=} \max_{f:A\to[0,1]} \Delta_\alpha(\mu_1\ f, \mu_2\ f)$$

The definition of α-distance quantifies universally over all unit-valued functions. The next lemma shows that for discrete distributions this definition is equivalent to an alternative definition in which quantification ranges only over Boolean-valued functions, i.e. those corresponding to characteristic functions of events.

Lemma 1. *For all distributions μ_1 and μ_2 over a discrete set A,*

$$\Delta_\alpha(\mu_1,\mu_2) = \max_{E\subseteq A} \Delta_\alpha(\mu_1\ \mathbb{1}_E, \mu_2\ \mathbb{1}_E)$$

An immediate consequence of Lemma 1 is that Δ_1 coincides with the standard notion of statistical distance, i.e.,

$$\Delta_1(\mu_1,\mu_2) = \max_{E\subseteq A} |\mu_1\ \mathbb{1}_E - \mu_2\ \mathbb{1}_E|$$

Differential privacy is a condition on the distance between the output distributions produced by a randomized algorithm. Namely, for a given metric on the input space, differential privacy requires that for any pair of inputs at distance at most 1, the probability that an algorithm outputs a particular value differs at most by a multiplicative factor $\exp(\epsilon)$. *Approximate* differential privacy relaxes this requirement by additionally allowing for an additive slack δ. The following definition captures these requirements in terms of α-distance; Lemma 1 establishes the equivalence to the original definition [16].

Definition 2 (Approximate differential privacy)**.** *Let d be a metric on A. A randomized algorithm $M : A \to \mathcal{D}(B)$ is (ϵ, δ)-differentially private (with respect to d) iff*

$$\forall a, a' \in A.\ d(a, a') \leq 1 \implies \Delta_{\exp(\epsilon)}(M\ a, M\ a') \leq \delta$$

Notice that $(\epsilon, 0)$-differential privacy corresponds to vanilla ϵ-differential privacy [13].

It is folklore that for discrete domains the definition of differential privacy is equivalent to its pointwise variant where one quantifies over characteristic functions of singleton sets rather than those of arbitrary sets; however, this equivalence breaks when considering approximate differential privacy [16]. The following lemma provides a way to establish bounds for α-distance (and hence for approximate differential privacy) in terms of characteristic functions of singleton sets. Note that the inequality is strict in general.

Lemma 2. *For all distributions μ_1 and μ_2 over a discrete set A,*

$$\Delta_\alpha(\mu_1, \mu_2) \leq \sum_{a \in A} \Delta_\alpha(\mu_1(a), \mu_2(a))$$

We conclude this section by stating some important properties of α-distance; these properties are used for reasoning about approximate lifting and proving the soundness of our logic. All properties are implicitly universally quantified.

Lemma 3 (Properties of α-distance)**.**

1. $0 \leq \Delta_\alpha(\mu_1, \mu_2) \leq 1$
2. $\Delta_\alpha(\mu, \mu) = 0$
3. $\Delta_\alpha(\mu_1, \mu_2) = \Delta_\alpha(\mu_2, \mu_1)$
4. $\Delta_{\alpha\alpha'}(\mu_1, \mu_3) \leq \alpha' \Delta_\alpha(\mu_1, \mu_2) + \Delta_{\alpha'}(\mu_2, \mu_3)$, *or else* $\Delta_{\alpha\alpha'}(\mu_1, \mu_3) \leq \Delta_\alpha(\mu_1, \mu_2) + \alpha\, \Delta_{\alpha'}(\mu_2, \mu_3)$
5. $\alpha \leq \alpha' \implies \Delta_{\alpha'}(\mu_1, \mu_2) \leq \Delta_\alpha(\mu_1, \mu_2)$
6. $\Delta_\alpha(\mathsf{bind}\ \mu_1\ M, \mathsf{bind}\ \mu_2\ M) \leq \Delta_\alpha(\mu_1, \mu_2)$

Most of the above properties are self-explanatory; we briefly highlight the most important ones. Property (4) generalizes the triangle inequality with appropriate skew factors; (5) states that α-distance is anti-monotonic with respect to α; (6) states that probabilistic computation does not increase the distance (which is a well-known fact for statistical distance).

4.2 Approximate Lifting of Relations to Distributions

The logic we present in the next section can be used to establish assertions about probabilistic programs w.r.t. pre- and post-conditions on states. In order to give meaning to these judgments, we need to interpret post-conditions as relations between distributions over states rather than as relations between states. To this end, we define the (α, δ)-lifting of a relation to distributions. Intuitively, two distributions $\mu_1 \in \mathcal{D}(A)$ and $\mu_2 \in \mathcal{D}(B)$ are related by the (α, δ)-lifting of $R \subseteq A \times B$, whenever there exists a distribution over $A \times B$ whose support is contained in R and whose first and second projections are at most at α-distance δ of μ_1 and μ_2, respectively.

Definition 3 (Lifting)**.** *Let $\alpha \in \mathbb{R}^{\geq 1}$ and $\delta \in [0, 1]$. The (α, δ)-lifting of a relation $R \subseteq A \times B$ is a relation over $\mathcal{D}(A) \times \mathcal{D}(B)$ defined as follows:*

$$\mu_1 \sim_R^{\alpha,\delta} \mu_2 \overset{def}{=} \exists \mu. \begin{cases} \mathsf{range}\ R\ \mu \\ \pi_1\ \mu \leq \mu_1 \ \wedge\ \pi_2\ \mu \leq \mu_2 \\ \Delta_\alpha(\pi_1\ \mu, \mu_1) \leq \delta \wedge \Delta_\alpha(\pi_2\ \mu, \mu_2) \leq \delta \end{cases}$$

We say that a distribution μ satisfying the above conditions is a witness *for the lifting.*

The notion of (α, δ)-lifting generalizes previous notions of liftings, such as the lifting from [20] which is obtained by taking $\alpha = 1$ and $\delta = 0$, and δ-lifting [12, 32] which is obtained by taking $\alpha = 1$. The next lemma shows that (α, δ)-lifting is monotonic w.r.t. the slack δ, the skew factor α, and the relation R. An immediate consequence is that for $\alpha > 1$, (α, δ)-lifting is more permissive than the previously proposed notions of lifting.

Lemma 4. *For all $1 \leq \alpha \leq \alpha'$ and $\delta \leq \delta'$, and relations $R \subseteq S$,*

$$\mu_1 \sim_R^{\alpha,\delta} \mu_2 \implies \mu_1 \sim_S^{\alpha',\delta'} \mu_2$$

We next present a fundamental property of (α, δ)-lifting, which is central to the applicability of apRHL to reason about α-distance (and hence differential privacy). Namely, two distributions related by the (α, δ)-lifting of R yield probabilities that are within α-distance δ when applied to R-related functions. We say that two functions $f : A \to [0, 1]$ and $g : B \to [0, 1]$ are related by a relation $R \subseteq A \times B$, and write $f =_R g$, iff for every $a \in A$ and $b \in B$, $R\ a\ b$ implies $f\ a = g\ b$.

Theorem 1 (Fundamental Property of Lifting)**.** *Let $\mu_1 \in \mathcal{D}(A)$, $\mu_2 \in \mathcal{D}(B)$, and $R \subseteq A \times B$. Then, for any two functions $f_1 : A \to [0, 1]$ and $f_2 : B \to [0, 1]$,*

$$\mu_1 \sim_R^{\alpha,\delta} \mu_2 \wedge f_1 =_R f_2 \implies \Delta_\alpha(\mu_1\ f_1, \mu_2\ f_2) \leq \delta$$

In particular, if $A = B$ and R is the identity relation ($\equiv$),

$$\mu_1 \sim_\equiv^{\alpha,\delta} \mu_2 \implies \Delta_\alpha(\mu_1, \mu_2) \leq \delta$$

Theorem 1 provides an interpretation of (α, δ)-lifting in terms of α-distance. Next we present two results that enable us to actually construct such liftings.

The first result is the converse of Theorem 1 for the special case of R being the identity relation: we prove that two distributions are related by the (α, δ)-lifting of the identity relation if their α-distance is smaller than δ. This result is used to prove the soundness of the logic rule for random assignments given in the next section.

Theorem 2. *Let μ_1 and μ_2 be distributions over a discrete set A. If $\Delta_\alpha(\mu_1, \mu_2) \leq \delta$, then $\mu_1 \sim_\equiv^{\alpha,\delta} \mu_2$.*

The proof is immediate by considering as a witness for the lifting the distribution with the following density function:

$$\mu(a, a') = \begin{cases} \min(\mu_1(a), \mu_2(a)) & \text{if } a = a' \\ 0 & \text{if } a \neq a' \end{cases}$$

The second result shows that (α, δ)-liftings compose. This result allows to derive a judgment relating two programs c_1 and c_2 by introducing an intermediate program c and proving the validity of judgments relating c_1 and c on one hand, and c and c_2 on the other. This approach is used in the examples of Section 6.2, and more extensively in cryptographic proofs, see e.g. [3].

Theorem 3. *Let μ_1, μ_2 and μ_3 be distributions over discrete sets A, B, and C, respectively. Let $R \subseteq A \times B$ and $S \subseteq B \times C$. For all $\alpha, \alpha' \in \mathbb{R}^{\geq 1}$ and $\delta, \delta' \in [0, 1]$,*

$$\mu_1 \sim_R^{\alpha,\delta} \mu_2 \wedge \mu_2 \sim_S^{\alpha',\delta'} \mu_3 \implies \mu_1 \sim_{R \circ S}^{\alpha\alpha',\delta''} \mu_3$$

where $\delta'' \overset{def}{=} \max(\delta + \alpha\ \delta', \delta' + \alpha'\ \delta)$ and $\circ$ denotes relation composition.

For the proof, let μ_R and μ_S be witnesses for the judgments on the left-hand side of the implication. Then, the distribution μ defined by the following density function is a witness for the lifting on the right-hand side:

$$\mu(a, c) = \sum_{\{b \in B \mid \mu_2(b) \neq 0\}} \frac{\mu_R(a, b)\ \mu_S(b, c)}{\mu_2(b)}$$

We conclude this section with an observation on (α, δ)-lifting for equivalence relations. Jonsson, Yi, and Larsen [20] show that for equivalence relations, their definition of lifting coincides with the more intuitive notion that requires the two distributions to yield equal probabilities on all equivalence classes. Formally, if R is an equivalence relation over a discrete set A, then

$$\mu_1 \sim_R^{1,0} \mu_2 \iff \forall a \in A.\ \mu_1\ \mathbb{1}_{[a]} = \mu_2\ \mathbb{1}_{[a]}$$

where $[a]$ denotes the R-equivalence class of $a \in A$. This characterization extends naturally to arbitrary α and $\delta = 0$:

$$\mu_1 \sim_R^{\alpha,0} \mu_2 \iff \forall a \in A.\ \Delta_\alpha(\mu_1\ \mathbb{1}_{[a]}, \mu_2\ \mathbb{1}_{[a]}) = 0$$

The characterization for arbitrary δ is more involved and is presented in the extended version.

5. Approximate Relational Hoare Logic

This section introduces the central component of CertiPriv, namely an approximate probabilistic relational Hoare logic that is used to establish privacy guarantees of programs. We first present the programming language and its semantics. We then define relational judgments and show that they generalize differential privacy. Finally, we define a proof system for deriving valid judgments.

5.1 Programming Language

CertiPriv supports reasoning about programs that are written in the typed, procedural, probabilistic imperative language pWHILE. Formally, the set of commands is defined inductively by the following clauses:

$\mathcal{I}$	$::=$	$\mathcal{V} \leftarrow \mathcal{E}$	assignment
	$\mid$	$\mathcal{V} \xleftarrow{\$} \mathcal{DE}$	random sampling
	$\mid$	$\mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C}$	conditional
	$\mid$	$\mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C}$	while loop
	$\mid$	$\mathcal{V} \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E})$	procedure call
	$\mid$	$\mathsf{assert}\ \mathcal{E}$	runtime assertion
$\mathcal{C}$	$::=$	skip	nop
	$\mid$	$\mathcal{I};\ \mathcal{C}$	sequence

Here, $\mathcal{V}$ is a set of variable identifiers, $\mathcal{P}$ is a set of procedure names[1], $\mathcal{E}$ is a set of expressions, and $\mathcal{DE}$ is a set of distribution expressions. The significant novelty of CertiPriv (compared to CertiCrypt), besides the addition of runtime assertions, is that the interpretation of distribution expressions may depend on the program state. This allows to express programs that sample from dynamically evolving probability distributions, such as the one presented in Section 2.

The semantics of programs is defined in two steps. First, we give an interpretation $[\![T]\!]$ to all object types T—these are types that are declared in CertiPriv programs, such as the graph type in the example in Section 2—and we define the set $\mathcal{M}$ of memories as the set of mappings from variables to values. Then, we implement dependently typed evaluators that give the semantics of an expression e of type T, a distribution expression μ of type T, and a command c, respectively, as functions of the following types:

$$[\![e]\!] : \mathcal{M} \to [\![T]\!] \qquad [\![\mu]\!] : \mathcal{M} \to \mathcal{D}([\![T]\!]) \qquad [\![c]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M})$$

Informally, the semantics of an expression e takes a memory and returns a value in $[\![T]\!]$, the semantics of a distribution expression μ takes a memory and returns a distribution over $[\![T]\!]$, and the semantics of a program c takes an initial memory and returns a distribution over final memories. The semantics of programs complies with the expected equations; Figure 3 provides an excerpt.

$$\begin{aligned}
[\![\mathsf{skip}]\!]\ m &= \mathsf{unit}\ m \\
[\![i;\ c]\!]\ m &= \mathsf{bind}\ ([\![i]\!]\ m)\ [\![c]\!] \\
[\![x \leftarrow e]\!]\ m &= \mathsf{unit}\ (m\,\{[\![e]\!]\ m/x\}) \\
[\![\mathsf{assert}\ e]\!]\ m &= \mathsf{if}\ [\![e]\!]\ m = \mathsf{true}\ \mathsf{then}\ (\mathsf{unit}\ m)\ \mathsf{else}\ \mu_0 \\
[\![x \xleftarrow{\$} \mu]\!]\ m &= \mathsf{bind}\ ([\![\mu]\!]\ m)\ (\lambda v.\ \mathsf{unit}\ (m\,\{v/x\})) \\
[\![\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!]\ m &= \begin{cases} [\![c_1]\!]\ m & \text{if } [\![e]\!]\ m = \mathsf{true} \\ [\![c_2]\!]\ m & \text{if } [\![e]\!]\ m = \mathsf{false} \end{cases} \\
[\![\mathsf{while}\ e\ \mathsf{do}\ c]\!]\ m &= \lambda f.\ \sup\ (\lambda n.\ [\![[\mathsf{while}\ e\ \mathsf{do}\ c]_n]\!]\ m\ f)
\end{aligned}$$

$$\text{where}\quad \begin{aligned} [\mathsf{while}\ e\ \mathsf{do}\ c]_0 &= \mathsf{skip} \\ [\mathsf{while}\ e\ \mathsf{do}\ c]_{n+1} &= \mathsf{if}\ e\ \mathsf{then}\ c;\ [\mathsf{while}\ e\ \mathsf{do}\ c]_n \end{aligned}$$

Figure 3. Semantics of pWHILE programs

5.2 Validity and Privacy

apRHL is an approximate probabilistic relational Hoare logic that supports reasoning about differentially private computations. Judgments in apRHL are of the form

$$c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi$$

where c_1 and c_2 are programs, Ψ and Φ are relations over memories, $\alpha \in \mathbb{R}^{\geq 1}$ is the skew, and $\delta \in [0, 1]$ is the slack. In our formalization we use a shallow embedding for logical assertions, allowing us to inherit the expressiveness of the Coq language when writing pre- and post-conditions.

An apRHL judgment is valid if for every pair of memories related by the pre-condition Ψ, the corresponding pair of output distributions is related by the (α, δ)-lifting of the post-condition Φ.

Definition 4 (Validity)**.** *A judgment $c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi$ is valid, written $\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi$, iff*

$$\forall m_1\, m_2.\ m_1\ \Psi\ m_2 \implies ([\![c_1]\!]\ m_1) \sim_\Phi^{\alpha,\delta} ([\![c_2]\!]\ m_2)$$

The following lemma is a direct consequence of the fundamental property of lifting (Theorem 1) applied to Definition 4. It shows that statements about programs derived using apRHL imply bounds on the α-distance of their output distributions.

Lemma 5. *If $\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi$, then for all memories m_1, m_2 and unit-valued functions $f_1, f_2 : \mathcal{M} \to [0, 1]$,*

$$m_1\ \Psi\ m_2 \wedge f_1 =_\Phi f_2 \implies \Delta_\alpha([\![c_1]\!]\ m_1\ f_1, [\![c_2]\!]\ m_2\ f_2) \leq \delta$$

The statement of Lemma 5 can be specialized to a statement about the differential privacy of programs.

Corollary 1. *Let d be a metric on $\mathcal{M}$ and Ψ an assertion expressing that $d(m_1, m_2) \leq 1$. If $\models c \sim_{\exp(\epsilon),\delta} c : \Psi \Rightarrow \equiv$, then c satisfies (ϵ, δ)-differential privacy.*

Corollary 1 is the central result for deriving differential privacy guarantees in apRHL. Using Theorem 2, one can prove the converse to Corollary 1, yielding a characterization of approximate differential privacy.

The logic apRHL can also be used to reason about more traditional information-flow properties, such as probabilistic noninterference. To see this, let Ψ be an arbitrary equivalence relation on initial states and let $\equiv$ be the identity relation on final states. A judgment $\models c \sim_{1,0} c : \Psi \Rightarrow \equiv$ entails that two initial states induce the same distribution of final states whenever they are related by Ψ. In particular, this implies that an adversary who can observe

[1] For the sake of readability, we omit procedure calls from most of the exposition; we keep them in the description of the language because we use them to describe the algorithm SMARTSUM in Fig. 9 and modularize its analysis.

(or even repeatedly sample) the output of c will only be able to determine the initial state up to its Ψ-equivalence class. In this way, Ψ can be used for expressing fine-grained notions of confidentiality, including probabilistic noninterference [31]. Our interpretation of apRHL judgments generalizes to arbitrary equivalence relations as post-conditions. In this way, one can capture adversaries that have only partial views on the system, as required for distributed differential privacy [5].

We finally show how apRHL can also be used for deriving generalized Lipschitz-conditions of probabilistic programs. As a first step, we show that valid apRHL judgments imply statements for input distributions that are related by (α, δ)-lifting.

Lemma 6. *If $\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi$, then for all distributions μ_1 and μ_2 of initial memories we have:*

$$\mu_1 \sim_{\Psi}^{\alpha',\delta'} \mu_2 \implies (\mathsf{bind}\ \mu_1\ [\![c_1]\!]\ ) \sim_{\Phi}^{\alpha\alpha',\delta+\delta'} (\mathsf{bind}\ \mu_2\ [\![c_2]\!])$$

By instantiating pre- and post-conditions in Lemma 6 to the identity relation on memories $\equiv$ and applying Theorems 1 and 2, one obtains the following generalized Lipschitz-condition for probabilistic programs on random inputs.

Corollary 2. *If $\models c_1 \sim_{\alpha,\delta} c_2 : \equiv \Rightarrow \equiv$, then*

$$\Delta_{\alpha'}(\mu_1, \mu_2) \leq \delta' \implies \Delta_{\alpha\alpha'}(\mathsf{bind}\ \mu_1\ [\![c_1]\!], \mathsf{bind}\ \mu_2\ [\![c_2]\!]) \leq \delta{+}\delta'$$

5.3 Logic

This section introduces a set of proof rules to support reasoning about the validity of apRHL judgments. In order to maximize flexibility and to allow the application of proof rules to be interleaved with other forms of reasoning, the soundness of each proof rule is proved individually as a Coq lemma. Nevertheless, we retain the usual presentation of the rules as a proof system.

We present the core apRHL rules in Figure 4; all rules generalize their pRHL counterpart, which can be recovered by setting $\alpha = 1$ and $\delta = 0$. (Any valid pRHL derivation admits an immediate translation into apRHL.) We begin by describing the rules corresponding to language constructs.

The [skip], [assert] and [assn] rules are direct transpositions of the pRHL rules. Rule [rand] states that for any two distribution expressions μ_1 and μ_2 of type A, the random assignments $x_1 \xleftarrow{\$} \mu_1$ and $x_2 \xleftarrow{\$} \mu_2$ are (α, δ)-related w.r.t. pre-condition Ψ and post-condition $x_1\langle 1\rangle = x_2\langle 2\rangle$, provided the α-distance between the distributions $[\![\mu_1]\!]\ m_1$ and $[\![\mu_2]\!]\ m_2$ is smaller than δ whenever m_1 and m_2 are related by Ψ. The soundness of rule [rand] follows from Theorem 2.

Rule [seq] has tight connections to composition theorems for differentially private algorithms, as further developed in Section 5.4.

Rule [cond] states that branching statements are (α, δ)-related w.r.t. pre-condition Ψ and post-condition Φ, provided that the pre-condition Ψ ensures that the guards of both statements are equivalent, and that the true and false branches are (α, δ)-related w.r.t. pre-conditions $\Psi \wedge b\langle 1\rangle$ and $\Psi \wedge \neg b\langle 1\rangle$, respectively.

The rule for loops may be best understood by taking $\delta = 0$. In this case, the rule [while] states that two bounded loops that execute in lockstep are $n \ln(\alpha)$-differentially private when their bodies at each iteration are $\ln(\alpha)$-differentially private and n is an upper bound for the number of iterations. The rule [while] is sufficient for proving differential privacy of examples like the k-median from [18], where the skew remains unchanged at each iteration. Other examples, like the ones discussed in the next section, require applying more sophisticated rules in which the skew and the slack may vary across iterations. For instance, the rule [gwhile] shown in Figure 5 allows for a finer-grained case analysis depending on a predicate P on program memories whose validity is preserved across iterations. Assume that when P does not hold, the iterations of each loop can be related with a privacy factor of $\alpha_1(i)$ in case P does not hold after their execution, and with a privacy factor of α_2 in case it does. Furthermore, assume that the iterations are observationally equivalent when P holds initially. Then, the two loops are related with a privacy factor of $(\prod_{i=1..n} \alpha_1(i))\ \alpha_2$. Intuitively, as long as P does not hold, the iterations of each loop are $\ln(\alpha_1(i))$-differentially private, while the single iteration where the validity of P may be established (this occurs necessarily at the same time in both loops) incurs an $\ln(\alpha_2)$ privacy penalty; the remaining iterations preserve P and do not add to the privacy bound.

We continue by explaining the structural rules given in Figure 4. The rule [case] allows one to perform a case analysis in the pre-condition of a judgment. The weakening rule [weak] generalizes the rule of consequence of (relational) Hoare logic by allowing to increase the skew and slack; its soundness follows from Lemma 4. The composition rule [comp] permits to structure proofs by introducing intermediate programs (as in the game-playing technique for cryptographic proofs [3]); its soundness follows from Theorem 3. Together with rule [transp], it yields a rule for the case when Ψ and Φ are partial equivalence relations which, specialized to $\alpha = \alpha' = 1$, reads:

$$\frac{\models c_1 \sim_{1,\delta} c_2 : \Psi \Rightarrow \Phi \qquad \models c_2 \sim_{1,\delta'} c_3 : \Psi \Rightarrow \Phi}{\models c_1 \sim_{1,\delta+\delta'} c_3 : \Psi \Rightarrow \Phi}$$

Finally, the [frame] rule allows one to strengthen the pre- and post-condition with an assertion Θ whose validity is preserved by executing the commands of the judgment. (In the figure, the notation $\times$ is used to denote the product of two distributions.)

5.4 Sequential and Parallel Composition Theorems

Composition theorems play an important role in the construction and analysis of differentially private mechanisms. A central result states that an ϵ-differentially private query followed by an ϵ'-differentially private query to the same dataset corresponds to a single $(\epsilon + \epsilon')$-differentially private query [24]. An important variant of this result deals with the case in which both queries access disjoint parts of the dataset. This so-called parallel composition of queries leads to a stronger, $\max\{\epsilon, \epsilon'\}$-privacy bound [24]. Both composition results admit a natural interpretation in apRHL, which we present below.

We begin by introducing additional notation that allows us to express independence of computations and observational capabilities of adversaries. For a set of variables $Z \subseteq \mathcal{V}$, we define the relations $\doteq_Z$ and $=_Z$ as follows:

$$\begin{array}{rcl} m =_Z m' & \overset{\text{def}}{=} & \forall y \in Z.\ m\ y = m'\ y \\ m \doteq_Z m' & \overset{\text{def}}{=} & \exists z \in Z.\ m =_{(Z \setminus \{z\})} m' \end{array}$$

Note that $m \doteq_Z m'$ corresponds to $d_Z(m, m') \leq 1$, where d_Z measures the number of variables in Z in which two memories differ, i.e. their Hamming distance. Using this notation, we can interpret judgments of the form

$$\models c \sim_{\exp(\epsilon),0} c : \doteq_X \Rightarrow =_Y$$

as a definition of a computation c on variables in X that is ϵ-differentially private w.r.t. adversaries that can only observe variables in Y. Similarly, we can interpret judgments of the form

$$\models c \sim_{\exp(\epsilon),0} c : \doteq_X \wedge =_{X'} \Rightarrow =_Y$$

as a definition of a family (indexed by X') of computations c on variables in X that are ϵ-differentially private w.r.t. adversaries that can only observe variables in Y. To emphasize the roles of the sets X, Y, and Y' we say that c is a computation from X to Y that is parameterized by Y'. Finally, we interpret premises of the form

$$\models c \sim_{1,0} c : =_X \Rightarrow =_X \qquad \models c \sim_{1,0} c : \doteq_X \Rightarrow \doteq_X$$

$$\frac{m_1\ \Psi\ m_2 \overset{\text{def}}{=} (m_1\,\{\llbracket e_1\rrbracket\ m_1/x_1\})\ \Phi\ (m_2\,\{\llbracket e_2\rrbracket\ m_2/x_2\})}{\models x_1 \leftarrow e_1 \sim_{1,0} x_2 \leftarrow e_2 : \Psi \Rightarrow \Phi}[\mathsf{assn}] \qquad \frac{\forall m_1\ m_2.\ m_1\ \Psi\ m_2 \implies \Delta_\alpha(\llbracket \mu_1\rrbracket\ m_1, \llbracket \mu_2\rrbracket\ m_2) \le \delta}{\models x_1 \xleftarrow{\$} \mu_1 \sim_{\alpha,\delta} x_2 \xleftarrow{\$} \mu_2 : \Psi \Rightarrow x_1\langle 1\rangle = x_2\langle 2\rangle}[\mathsf{rand}]$$

$$\frac{\Psi \implies b\langle 1\rangle \equiv b'\langle 2\rangle}{\models \mathsf{assert}\ b \sim_{1,0} \mathsf{assert}\ b' : \Psi \Rightarrow \Psi \wedge b\langle 1\rangle}[\mathsf{assert}] \qquad \frac{\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi' \qquad \models c_1' \sim_{\alpha',\delta'} c_2' : \Phi' \Rightarrow \Phi}{\models c_1; c_1' \sim_{\alpha\alpha',\delta+\delta'} c_2; c_2' : \Psi \Rightarrow \Phi}[\mathsf{seq}]$$

$$\frac{}{\models \mathsf{skip} \sim_{1,0} \mathsf{skip} : \Psi \Rightarrow \Psi}[\mathsf{skip}] \qquad \frac{\models c_1 \sim_{\alpha,\delta} c_1' : \Psi \wedge b\langle 1\rangle \Rightarrow \Phi \quad \models c_2 \sim_{\alpha,\delta} c_2' : \Psi \wedge \neg b\langle 1\rangle \Rightarrow \Phi \qquad \Psi \implies b\langle 1\rangle \equiv b'\langle 2\rangle}{\models \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \sim_{\alpha,\delta} \mathsf{if}\ b'\ \mathsf{then}\ c_1'\ \mathsf{else}\ c_2' : \Psi \Rightarrow \Phi}[\mathsf{cond}]$$

$$\frac{\models c \sim_{\alpha,\delta} c' : \Psi \wedge b\langle 1\rangle \wedge b'\langle 2\rangle \Rightarrow \Psi \wedge b\langle 1\rangle \equiv b'\langle 2\rangle \quad \forall m_1\ m_2.\ m_1\ \Psi\ m_2 \implies \llbracket \mathsf{while}\ b\ \mathsf{do}\ c\rrbracket\ m_1 = \llbracket [\mathsf{while}\ b\ \mathsf{do}\ c]_n\rrbracket\ m_1}{\models \mathsf{while}\ b\ \mathsf{do}\ c \sim_{\alpha^n,n\delta} \mathsf{while}\ b'\ \mathsf{do}\ c' : \Psi \wedge b\langle 1\rangle \equiv b'\langle 2\rangle \Rightarrow \Psi \wedge \neg b\langle 1\rangle \wedge \neg b'\langle 2\rangle}[\mathsf{while}]$$

$$\frac{\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \wedge \Theta \Rightarrow \Phi \qquad \models c_1 \sim_{\alpha,\delta} c_2 : \Psi \wedge \neg\Theta \Rightarrow \Phi}{\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi}[\mathsf{case}] \qquad \frac{\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi \qquad \models c_2 \sim_{\alpha,\delta'} c_3 : \Psi' \Rightarrow \Phi'}{\models c_1 \sim_{\alpha\alpha',\max(\delta+\alpha\ \delta',\delta'+\alpha'\ \delta)} c_3 : \Psi \circ \Psi' \Rightarrow \Phi \circ \Phi'}[\mathsf{comp}]$$

$$\frac{\models c_1 \sim_{\alpha',\delta'} c_2 : \Psi' \Rightarrow \Phi' \quad \Psi \Rightarrow \Psi' \quad \Phi' \Rightarrow \Phi \quad \alpha' \le \alpha \quad \delta' \le \delta}{\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi}[\mathsf{weak}] \qquad \frac{\models c_2 \sim_{\alpha,\delta} c_1 : \Psi^{-1} \Rightarrow \Phi^{-1}}{\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi}[\mathsf{transp}]$$

$$\frac{\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi \qquad \forall m_1\ m_2.\ m_1\ \Theta\ m_2 \implies \mathsf{range}\ \Theta\ (\llbracket c_1\rrbracket\ m_1 \times \llbracket c_2\rrbracket\ m_2)}{\models c_1 \sim_{\alpha,\delta} c_2 : \Psi \wedge \Theta \Rightarrow \Phi \wedge \Theta}[\mathsf{frame}]$$

Figure 4. Core proof rules of the approximate relational Hoare logic

$$\frac{\begin{array}{c}\Phi \implies (b_1\langle 1\rangle \equiv b_2\langle 2\rangle \wedge P_1\langle 1\rangle \equiv P_2\langle 2\rangle \wedge i\langle 1\rangle = i\langle 2\rangle)\\ \forall m_1\ m_2.\ m_1\ \Phi\ m_2 \implies \llbracket \mathsf{while}\ b_1\ \mathsf{do}\ c_1\rrbracket\ m_1 = \llbracket [\mathsf{while}\ b_1\ \mathsf{do}\ c_1]_n\rrbracket\ m_1\\ \models c_1;\ \mathsf{assert}\ (\neg P_1) \sim_{\alpha_1(j),0} c_2;\ \mathsf{assert}\ (\neg P_2) : \Phi \wedge b_1\langle 1\rangle \wedge i\langle 1\rangle = j\ \wedge \neg P_1\langle 1\rangle \Rightarrow \Phi \wedge i\langle 1\rangle = j+1\\ \models c_1;\ \mathsf{assert}\ (P_1) \sim_{\alpha_2,0} c_2;\ \mathsf{assert}\ (P_2) : \Phi \wedge b_1\langle 1\rangle \wedge i\langle 1\rangle = j \wedge \neg P_1\langle 1\rangle \Rightarrow \Phi \wedge i\langle 1\rangle = j+1\\ \models c_1 \sim_{1,0} c_2 : \Phi \wedge b_1\langle 1\rangle \wedge i\langle 1\rangle = j \wedge P_1\langle 1\rangle \Rightarrow \Phi \wedge i\langle 1\rangle = j+1 \wedge P_1\langle 1\rangle\end{array}}{\models \mathsf{while}\ b_1\ \mathsf{do}\ c_1 \sim_{(\prod_{i=a}^{a+n} \alpha_1(i))\times\alpha_2,0} \mathsf{while}\ b_2\ \mathsf{do}\ c_2 : \Phi \wedge i\langle 1\rangle = a \Rightarrow \Phi \wedge \neg b_1\langle 1\rangle}[\mathsf{gwhile}]$$

Figure 5. Generalized rule for loops

as a statement that c does not modify variables in X. Note that this reading of premises is sound, but stronger than the actual semantics.

The sequential composition theorem is a direct application of rule [seq] and is captured by the rule:

$$\frac{\begin{array}{c}\models c \sim_{\exp(\epsilon),0} c : \doteq_X \Rightarrow (=_Y \wedge \doteq_X)\\ \models c' \sim_{\exp(\epsilon'),0} c' : (=_Y \wedge \doteq_X) \Rightarrow =_{(Y\cup Y')}\end{array}}{\models c; c' \sim_{\exp(\epsilon+\epsilon'),0} c; c' : \doteq_X \Rightarrow =_{(Y\cup Y')}}$$

With the intuitive reading introduced above, the rule states that the composition of an ϵ-differentially private computation c from X to Y, with a parameterized ϵ-differentially private computation c' from X to Y' is an $(\epsilon+\epsilon')$-differentially private computation $c; c'$ from X to $Y \cup Y'$, provided that c' does not modify variables in Y, and c does not modify variables in X.

The parallel composition theorem is captured by the rule:

$$\frac{\begin{array}{l}\models c \sim_{\exp(\epsilon),0} c : (\doteq_X \wedge =_{X'}) \Rightarrow =_{(Y\cup X')}\\ \models c \sim_{1,0} c : (=_X \wedge \doteq_{X'}) \Rightarrow (=_Y \wedge \doteq_{X'})\\ \models c' \sim_{\exp(\epsilon'),0} c' : (=_Y \wedge \doteq_{X'}) \Rightarrow =_{(Y\cup Y')}\\ \models c' \sim_{1,0} c' : =_{(Y\cup X')} \Rightarrow =_{(Y\cup Y')} \qquad X \cap X' = \emptyset\end{array}}{\models c; c' \sim_{\exp(\max(\epsilon,\epsilon')),0} c; c' : \doteq_{X\cup X'} \Rightarrow =_{(Y\cup Y')}}$$

With the intuitive reading introduced above, the rule states that the composition of an ϵ-differentially private computation c from X to Y, with a parameterized ϵ-differentially private computation c' from X' to Y', where X' is disjoint from X, is a $\max(\epsilon,\epsilon')$-differentially private computation $c; c'$ from $X \cup X'$ to $Y \cup Y'$. As a prerequisite we require that c does not modify variables in X' and is non-interfering w.r.t. input variables X and output variables Y, and that c' does not modify variables in Y and is non-interfering w.r.t. input variables X' and output variables Y'.

6. Case Studies

We illustrate the versatility of our framework by formalizing two prominent mechanisms, namely the Laplacian and the Exponential mechanisms, and proving their correctness from first principles. We then apply these mechanisms to prove differential privacy for several streaming algorithms. Finally, we detail the proof of differential privacy of the Minimum Vertex Cover algorithm introduced in Section 2.

6.1 Exponential and Laplacian Mechanisms

Many algorithms for statistics and data mining are numeric, i.e. they return (approximations of) real numbers. The Laplacian mechanism of Dwork et al. [17] is a fundamental tool for making such computations differentially private. This is achieved by perturbing the algorithm's true output with noise drawn from a Laplace distribution. The density function at x of the Laplace distribution centered around r with scale factor σ is proportional to

$$\exp\left(-\frac{|x-r|}{\sigma}\right)$$

$$\frac{m_1\ \Psi\ m_2 \implies |[\![r]\!]\ m_1 - [\![r]\!]\ m_2| \leq k \qquad \exp(\epsilon) \leq \alpha}{\models x \xleftarrow{\$} \mathcal{L}(r, \frac{k}{\epsilon}) \sim_{\alpha,0} y \xleftarrow{\$} \mathcal{L}(r, \frac{k}{\epsilon}) : \Psi \Rightarrow x\langle 1\rangle = y\langle 2\rangle}[\mathsf{lap}] \quad \frac{m_1\ \Psi\ m_2 \implies d([\![a]\!]\ m_1, [\![a]\!]\ m_2) \leq k \qquad \exp(2k\mathbf{S}_s\epsilon) \leq \alpha}{\models x \xleftarrow{\$} \mathcal{E}^{\epsilon}_{s,\mu}(a) \sim_{\alpha,0} y \xleftarrow{\$} \mathcal{E}^{\epsilon}_{s,\mu}(a) : \Psi \Rightarrow x\langle 1\rangle = y\langle 2\rangle}[\mathsf{exp}]$$

Figure 6. Rules for the Laplacian and Exponential mechanisms

To transform a deterministic computation $f\colon A \to \mathbb{R}$ into a differentially private computation, one needs to set r to the true output of the computation and choose σ (i.e. the amount of noise) according to the *sensitivity* of f. Informally, the sensitivity is a Lipschitz-parameter that captures how far apart f maps nearby inputs. Formally, the sensitivity $\mathbf{S}_f$ is defined relative to a metric d on A as follows:

$$\mathbf{S}_f \stackrel{\text{def}}{=} \max_{a,a' | d(a,a') \leq 1} |f\ a - f\ a'|$$

The justification for the Laplacian mechanism is a result that states that for a function $f : A \to \mathbb{R}$, the randomized algorithm that on input a returns a value sampled from the Laplacian distribution centered around $f(a)$ with scale factor $\sigma = \mathbf{S}_f/\epsilon$ is ϵ-differentially private [17].

One limitation of the Laplacian mechanism is that it is confined to numerical algorithms. The Exponential mechanism [23] is a general mechanism for building differentially private algorithms with arbitrary (but discrete) output domains. The Exponential mechanism takes as input a base distribution μ on a set B, and a scoring function $s : A \times B \to \mathbb{R}^{\geq 0}$; intuitively, values b maximizing $s(a, b)$ are the most appealing output for an input a. The Exponential mechanism is a randomized algorithm that takes a value $a \in A$ and returns a value $b \in B$ that approximately maximizes the score $s(a, b)$, where the quality of the approximation is determined by a parameter $\epsilon > 0$. Formally, the Exponential mechanism $\mathcal{E}^{\epsilon}_{s,\mu}$ maps every element in A to a distribution in B whose density function at b is given by:

$$\mathcal{E}^{\epsilon}_{s,\mu}(a)\ b = \frac{\exp(\epsilon\ s(a,b))\ (\mu\ b)}{\sum_{b \in B} \exp(\epsilon\ s(a,b))\ (\mu\ b)}$$

The definition implicitly assumes that the sum in the denominator is bounded for all $a \in A$. McSherry and Talwar [23] show that $\mathcal{E}^{\epsilon}_{s,\mu}$ is $2\epsilon\mathbf{S}_s$-differentially private, where $\mathbf{S}_s$ is the maximum sensitivity of s w.r.t. a, for all b.

In our proofs, the Exponential and Laplacian mechanisms are defined as instances of a general construction $(\cdot)^\sharp$ that takes as input a function $f : A \to B \to \mathbb{R}^{\geq 0}$ and returns a function $f^\sharp : A \to \mathcal{D}(B)$ such that for every $a \in A$ the density function of $f^\sharp\ a$ at b is given by:

$$f^\sharp\ a\ b = \frac{f\ a\ b}{\sum_{b \in B} f\ a\ b}$$

We derive the correctness of the Laplacian and Exponential mechanisms as a consequence of the following lemma.

Lemma 7. *Let B be a discrete set and consider a function $f : A \to B \to \mathbb{R}^{\geq 0}$ such that $f^\sharp$ is well defined. Let $a, a' \in A$, $\gamma \geq 0$ be such for all b, $f(a, b) \leq \gamma\ f(a', b)$ and $f(a', b) \leq \gamma\ f(a, b)$. Then,*

$$\Delta_{\gamma^2}(f^\sharp\ a, f^\sharp\ a') = 0$$

If moreover $\sum_{b \in B} f\ a\ b = \sum_{b \in B} f\ a'\ b$, then

$$\Delta_{\gamma}(f^\sharp\ a, f^\sharp\ a') = 0$$

Using the construction $(\cdot)^\sharp$ defined above, the Exponential mechanism for a scoring function s, base distribution μ and scale factor ϵ is defined as

$$\mathcal{E}^{\epsilon}_{s,\mu} \stackrel{\text{def}}{=} (\lambda a\ b.\ \exp(\epsilon\ s(a,b))\ (\mu\ b))^\sharp$$

whereas the Laplacian mechanism with mean value r and scale factor σ is defined as

$$\mathcal{L}(r,\sigma) \stackrel{\text{def}}{=} \left(\lambda a\ b.\ \exp\left(-\frac{|b-a|}{\sigma}\right)\right)^\sharp r$$

The privacy guarantees for the Exponential and the Laplacian mechanisms are stated as the rules [lap] and [exp] in Figure 6; their soundness is a corollary of Lemma 7 above.

Observe that the premise of rule [lap] requires to prove that the values around which the mechanism is centered are within distance k. This is the case when these values are computed by a k-sensitive function starting from adjacent inputs, which corresponds to the usual interpretation of the guarantees provided by the Laplacian mechanism [17].

As a further illustration of the expressive power of CertiPriv, we have also defined a Laplacian mechanism $\mathcal{L}^n$ for lists; given $\sigma \in \mathbb{R}^+$ and a vector $a \in \mathbb{R}^n$, the mechanism $\mathcal{L}^n$ outputs a vector in $\mathbb{R}^n$ whose i-th component is drawn from distribution $\mathcal{L}(a[i], \sigma)$. More formally, we have proved the soundness of the following rule

$$\frac{m_1\ \Psi\ m_2 \implies \sum_{1 \leq i \leq n} |[\![a[i]]\!]\ m_1 - [\![a[i]]\!]\ m_2| \leq k}{\models x \xleftarrow{\$} \mathcal{L}^n(a, \frac{k}{\epsilon}) \sim_{\exp(\epsilon),0} y \xleftarrow{\$} \mathcal{L}^n(a, \frac{k}{\epsilon}) : \Psi \Rightarrow x\langle 1\rangle = y\langle 2\rangle}$$

which we refer to as [lap*].

6.2 Statistics over Streams

In this section we present an analysis of algorithms for computing private and continual statistics in a data stream [8]. As in [8], we focus on algorithms for private summing and counting. More sophisticated algorithms, e.g. computing heavy hitters in a data stream, can be built using sums and counters as primitive operations and inherit their privacy and utility guarantees.

We consider streams of elements in a bounded subset $D \subseteq \mathbb{R}$, i.e. with $|x - y| \leq b$ for all $x, y \in D$. This setting is slightly more general than the one considered by Chan et al. [8], where only streams over $\{0, 1\}$ are considered. On the algorithmic side, the generalization to bounded domains is immediate; for the privacy analysis, however, one needs to take the bound b into account because it conditions the sensitivity of computations. This requires a careful definition of metrics and propagation of bounds, which is supported by CertiPriv.

Although in our implementation we formalize streams as finite lists, we use array-notation in the exposition for the sake of readability. Given an array a of n elements in D, the goal is to release, for every point in time $0 \leq j < n$ the aggregate sum $c[j] = \sum_{i=0}^{j} a[i]$ in a privacy-preserving manner. As observed in [8], there are two immediate solutions to the problem. The first is to maintain an exact aggregate sum $c[j]$ and output at each iteration a curated version $\overline{c}[j] \xleftarrow{\$} \mathcal{L}(c[j], b/\epsilon)$ of that sum. The second solution is to maintain and output a noisy aggregate sum $\tilde{c}[j]$, which is updated at iteration $j + 1$ according to

$$\overline{a}[j+1] \xleftarrow{\$} \mathcal{L}(a[j+1], b/\epsilon);\ \tilde{c}[j+1] \leftarrow \tilde{c}[j] + \overline{a}[j+1]$$

The stream $\overline{c}[0] \cdots \overline{c}[n-1]$ offers weak, $n\epsilon$-differential privacy, because every element of a may appear in n different elements of $\overline{c}$, each with independent noise. However, each $\overline{c}[j]$ offers good accuracy because noise is added only once. In contrast, the stream $\tilde{c}[0] \cdots \tilde{c}[n-1]$ offers improved, ϵ-differential privacy, because each element of a appears only in one ϵ-differentially private query. However, as shown in [8], the sum $\tilde{c}[j]$ yields poor accuracy because noise is added j times during its computation.

One solution proposed by Chan et al. [8] is a combination of both basic methods of releasing partial sums that achieves a good compromise between privacy and accuracy. The idea is to split the stream a into chunks of length q, where the less accurate (but more private) method is used to compute the sum within the current chunk, and the more accurate (but less private) method is used to compute summaries of previous chunks. Formally, let $s_t = \sum_{i=0}^{q-1} a[t\ q + i]$ be the sum over the t-th chunk of a and let $\overline{s}_t \xleftarrow{\$} \mathcal{L}(s_t, b/\epsilon)$ be the corresponding noisy version. Then, for each $j = qr + k$, with $k < q$, we compute

$$\hat{c}[j] = \sum_{t=0}^{r-1} \overline{s}_t + \sum_{i=0}^{k} \overline{a}[qr + i]$$

The sequence $\hat{c}[0] \cdots \hat{c}[n-1]$ offers 2ϵ-differential privacy, intuitively because each element of a is accessed twice during computation. Moreover, $\hat{c}[j]$ also offers improved accuracy over $\tilde{c}[j]$ because noise is added only $r + k$ times rather than $j = qr + k$ times.

We will now turn the above informal security analysis into a formal analysis of program code. The code for computing $\overline{s}_t$ is given as the function PartialSum in Figure 7, the code for computing $\overline{c}$ is given as the function PartialSum' in Figure 8, and the code for computing $\hat{c}$ is given as the function SmartSum in Figure 9 (we omit the code for computing $\tilde{c}$ and the proof of its privacy bound). We next sketch the key steps in our proofs of differential privacy bounds for each of these algorithms. For all of our examples, we use the pre-condition

$$\begin{aligned}\Psi \overset{\text{def}}{=}\ & \mathsf{length}(a\langle 1\rangle) = \mathsf{length}(a\langle 2\rangle) \wedge a\langle 1\rangle \doteq a\langle 2\rangle \wedge \\ & \forall i.\ 0 \le i < \mathsf{length}(a\langle 1\rangle) \implies |a[i]\langle 1\rangle - a[i]\langle 2\rangle| \le b\end{aligned}$$

which relates two lists $a\langle 1\rangle$ and $a\langle 2\rangle$ whenever they have the same length, differ in at most one element, and the distance between the elements at the same position at each array is upper-bounded by b.

PartialSum The proof of differential privacy of PartialSum proceeds in two key steps. First, we prove (using the pRHL fragment of apRHL) that

$$\models c_{1-5} \sim_{1,0} c_{1-5} : \Psi \Rightarrow |s\langle 1\rangle - s\langle 2\rangle| \le b$$

where c_{1-5} corresponds to the code in lines 1-5 in Figure 7, i.e. the initialization and the loop. We apply the rule [lap] that gives a bound for the privacy guarantee achieved by the Laplacian mechanism (see Figure 6) to $c_6 = s \xleftarrow{\$} \mathcal{L}(s, b/\epsilon)$ (the instruction in line 6) and derive

$$\models c_6 \sim_{\exp(\epsilon),0} c_6 : |s\langle 1\rangle - s\langle 2\rangle| \le b \Rightarrow s\langle 1\rangle = s\langle 2\rangle$$

Using the rule for sequential composition, applied to c_{1-5} and c_6, we derive the following statement about PartialSum, which implies that its output s is ϵ-differentially private.

$$\models \text{PartialSum}(a) \sim_{\exp(\epsilon),0} \text{PartialSum}(a) : \Psi \Rightarrow s\langle 1\rangle = s\langle 2\rangle$$

PartialSum' Our implementation of PartialSum' in Figure 7 differs slightly from the description given above in that we first add noise to the entire stream (line 1), before computing the partial sums of the noisy stream (lines 2-6). This modification allows us to take advantage of the proof rule for the Laplacian mechanism on lists. By merging the addition of noise into the loop, our two-pass implementation can be turned into an observationally equivalent one-pass implementation suitable for processing streams of data.

```
function PartialSum($a$)
1   $s \leftarrow 0; i \leftarrow 0$;
2   while $i <$ length($a$) do
3       $s \leftarrow s + a[i]$;
4       $i \leftarrow i + 1$;
5   end;
6   $s \leftarrow \mathcal{L}(s, b/\epsilon)$
```

Figure 7. A simple ϵ-differentially private algorithm for summing over lists

```
function PartialSum'($a$)
1   $\overline{a} \xleftarrow{\$} \mathcal{L}^n(a, b/\epsilon)$;
2   $s[0] \leftarrow \overline{a}[0]; i \leftarrow 1$;
3   while $i <$ length($a$) do
4       $s[i] \leftarrow s[i-1] + \overline{a}[i]$;
5       $i \leftarrow i + 1$;
6   end
```

Figure 8. An ϵ-differentially private algorithm for partial sums over lists

```
function SmartSum($a, q$)
1   $i \leftarrow 0; c \leftarrow 0$;
2   while $i <$ length($a$)$/q$ do
3       $b \leftarrow$ PartialSum($a[iq..i(q+1)-1]$);
4       $x \leftarrow$ PartialSum'($a[iq..i(q+1)-1]$);
5       $s \leftarrow$ OffsetCopy($s, x, c, iq, q$);
6       $c \leftarrow c + b$;
7       $i \leftarrow i + 1$;
8   end
```

Figure 9. A smart 2ϵ-differentially private algorithm for partial sums over lists

The proof of privacy for PartialSum' proceeds in the following basic steps. First, we apply the rule [lap*] to the random assignment in line 1 (noted as c_1) of PartialSum'. We obtain

$$\models c_1 \sim_{\exp(\epsilon),0} c_1 : \Psi \Rightarrow \overline{a}\langle 1\rangle = \overline{a}\langle 2\rangle$$

i.e. the output $\overline{a}$ is ϵ-differentially private at this point. For lines 2-6 (denoted by c_{2-6}), we prove (using the pRHL fragment of apRHL) that

$$\models c_{2-6} \sim_{1,0} c_{2-6} : \overline{a}\langle 1\rangle = \overline{a}\langle 2\rangle \Rightarrow s\langle 1\rangle = s\langle 2\rangle$$

This is straightforward because of the equality appearing in the precondition; this result can be derived using the apRHL rules, but is also an immediate consequence of the preservation of α-distance by probabilistic computations (see Lemma 3).

Finally, we apply the rule for sequential composition to c_1 and c_{2-6} and obtain

$$\models \text{PartialSum'}(a) \sim_{\exp(\epsilon),0} \text{PartialSum'}(a) : \Psi \Rightarrow s\langle 1\rangle = s\langle 2\rangle$$

which implies that the output s of PartialSum' is ϵ-differentially private.

SmartSum Our implementation of the smart private sum in Figure 9 makes use of PartialSum and PartialSum' as building blocks, which enables us to reuse the above proofs.

In addition, our implementation makes use of a procedure OffsetCopy that given two lists s and x, a constant c and non-negative integers i, q, returns a list which is identical to s, but where the entries $s[i] \cdots s[i + (q-1)]$ are replaced by the first q elements of x, plus a constant offset c, i.e. $s[i+j] = x[j]+c$ for $0 <= i < q$.

We obtain
$\models s \leftarrow \textsc{OffsetCopy}(s,x,c,i,q) \sim_{1,0} s \leftarrow \textsc{OffsetCopy}(s,x,c,i,q) : =_{\{s,x,c,i,q\}} \Rightarrow s\langle 1\rangle = s\langle 2\rangle$
We combine this result with the judgments derived for PARTIALSUM and PARTIALSUM' using the rule for sequential composition, obtaining

$$\models c_{4-7} \sim_{\exp(2\epsilon),0} c_{4-7} : \Psi \Rightarrow s\langle 1\rangle = s\langle 2\rangle$$

where c_{4-7} denotes the body of the loop in lines 4-7. To conclude, we apply the rule for while loops in Fig. 5 with $\alpha_1(i) = 1$ and $\alpha_2 = \exp(2\epsilon)$. This instantiation of the rule states that a loop that is non-interfering in all but one iteration is 2ϵ-differentially private, if the interfering loop iteration is 2ϵ-differentially private. More technically, the existence of a single interfering iteration is built into the rule using a pair of stable events for each command. In our case, the critical iteration corresponds to the one in which the chunk contains the position in which the two lists differ.

6.3 Minimum Vertex Cover

We conclude this section with a more detailed account of the proof of differential privacy of the Minimum Vertex Cover approximation algorithm of Section 2. The main step of the proof is an application of the rule for loops in Fig. 5 with parameters

$$\alpha_1(i) = \exp\left(\frac{2}{(n-i)w_i}\right) \qquad \alpha_2 = \exp\left(\frac{\epsilon}{4}\right),$$

the following invariant Φ

$$\begin{array}{l}(t \in \pi\langle 1\rangle \vee u \in \pi\langle 1\rangle \implies E\langle 1\rangle = E\langle 2\rangle) \wedge \\ (t \notin \pi\langle 1\rangle \wedge u \notin \pi\langle 1\rangle \implies E\langle 2\rangle = E\langle 1\rangle \cup \{(t,u)\}) \wedge \\ V\langle 1\rangle = V\langle 2\rangle \wedge \pi\langle 1\rangle = \pi\langle 2\rangle,\end{array}$$

and stable properties $P_1 = P_2 = t \in \pi \vee u \in \pi$.

The first and second equivalences appearing in the premises of the rule are of the form:

$$\models c_1; \mathsf{assert}\ P \sim_{\alpha,0} c_2; \mathsf{assert}\ P : \Psi \Rightarrow \Phi$$

For each of them, we first hoist the assertion immediately after the random assignment. At this point the expression in the assertions becomes $(t, u \notin (v :: \pi))$ in the case of the first premise and $(t \in (v :: \pi) \vee u \in (v :: \pi))$ in the case of the second. We then compute the weakest pre-condition of the assignments that now follow the assertions. The resulting judgments simplify, after applying the [weak] and [frame] rules, to judgments of the form

$$\models c \sim_{\alpha,0} c : \Psi \Rightarrow v\langle 1\rangle = v\langle 2\rangle$$

where

$$\begin{array}{rl}\Psi \overset{\text{def}}{=} & E\langle 2\rangle = E\langle 1\rangle \cup \{(t,u)\} \wedge V\langle 1\rangle = V\langle 2\rangle \wedge t, u \notin \pi \wedge \\ & i\langle 1\rangle = i\langle 2\rangle = j \wedge \pi\langle 1\rangle = \pi\langle 2\rangle\end{array}$$

For the first premise we have $\alpha = \alpha_1(j)$ and

$$c = v \overset{\$}{\leftarrow} \mathsf{choose}(V, E, \epsilon, n, i);\ \mathsf{assert}\ (t, u \notin (v :: \pi))$$

whereas for the second premise we have $\alpha = \alpha_2$ and

$$c = v \overset{\$}{\leftarrow} \mathsf{choose}(V, E, \epsilon, n, i);\ \mathsf{assert}\ (t \in (v :: \pi) \vee u \in (v :: \pi))$$

To establish the validity of both judgments, we cast the code for c as a random assignment where v is sampled from the interpretation of $\mathsf{choose}(V, E, \epsilon, n, i)$ restricted to v satisfying the condition on the assertion. In the first case, the restriction amounts to $v \neq u, t$ whereas in the second it amounts to $v = t \vee v = u$. For each one of these cases, we apply the rule for random assignments and are thus left to prove that the α-distance of the corresponding distributions is null. In view of Lemma 2, this in turn amounts to verifying for each element x in the support of the distribution that the ratio between the probability of v being equal to x in the left-hand side (resp. right-hand side) program and the right-hand side (resp. left-hand side) program is bounded by α, which directly translates into the inequalities appearing in Section 2. Technically, these inequalities are proved by appealing to a variant of Lemma 7.

The proof in apRHL yields a bound of $5\epsilon/4$ rather than the ϵ bound from [18]. This difference is due to the symmetric nature of our logic. We believe that the optimal bound can be proved in apRHL at the cost of a more complicated proof by using rule [comp] to introduce intermediate programs or, more elegantly, by using an asymmetric version of apRHL. See the appendix for a discussion on what it would take to build an asymmetric logic and how it could be used to improve the privacy bound from $5\epsilon/4$ to just ϵ.

7. Related Work

Our work builds upon program verification techniques, and in particular (probabilistic and relational) program logics, to reason about differential privacy. We briefly review relevant work in these areas.

Differential privacy There is a vast body of work on differential privacy. We refer to recent overviews, see e.g. [14, 15], for an account of some of the latest developments in the field, and focus on language-based approaches to differential privacy. The Privacy Integrated Queries (PINQ) platform [24] supports reasoning about the privacy guarantees of programs in a simple SQL-like language. The reasoning is based on the sensitivity of basic queries such as `Select` and `GroupBy`, the differential privacy of building blocks such as `NoisySum` and `NoisyAvg`, and meta-theorems for their sequential and parallel composition. AIRAVAT [30] leverages these building blocks for distributed computations based on MapReduce.

The linear type system of [29] extends sensitivity analysis to a higher-order functional language. By using a suitable choice of metric and probability monads, the type system also supports reasoning about probabilistic, differentially private computations. As in PINQ, the soundness of the type system makes use of known composition theorems and relies on assumptions about the sensitivity/differential privacy of nontrivial building blocks, such as arithmetic operations, conditional swap operations, or the Laplacian mechanism. While the type system can handle functional data structures, it does not allow for analyzing programs with conditional branching. Work on the automatic derivation of sensitivity properties of imperative programs [9] addresses this problem and can (in conjunction with the Laplacian mechanism) be used to derive differential privacy guarantees of programs with control flow. Although this approach supports reasoning about probabilistic computations, the reasoning is restricted to Lipschitz-conditions.

In contrast to [9, 24, 29], CertiPriv supports reasoning about differential privacy guarantees from first principles. In particular, CertiPriv enabled us to prove (rather than to assume) the correctness of Laplacian and Exponential mechanisms, and the differential privacy of complex interleavings of (not necessarily differentially private) probabilistic computations.

A recent approach considers the verification of privacy properties based on I/O-automata [35]. There, the focus lies on the verification of the correct use of differentially private sanitization mechanisms within interactive systems, where the effect of a sanitization mechanism is soundly abstracted using a single, idealized transition.

An early approach to quantitative confidentiality analysis [26] uses the distance of output distributions to quantify information flow. Their measure is closely related to $(0, \delta)$-approximate differential privacy, which can be reasoned about in CertiPriv. More recent approaches to quantitative information-flow focus on measures of confidentiality based on information-theoretic entropy. Techniques for code-based structural reasoning about these measures are developed in [10]. For an overview and a discussion of the relationship between entropy-based measures of confidentiality and differential privacy, see [2].

Probabilistic and relational program verification Program logics have a long tradition and have been used effectively to reason about functional correctness of programs. In contrast, privacy is a 2-safety property [11, 33], that is, a (universally quantified) property about two runs of a program. There have been several proposals for applying program logics to 2-safety, but these proposals are confined to deterministic programs and impractical.

A seminal paper [7] develops a relational Hoare logic (RHL) for a core imperative programming language and shows how it can be used to reason about information flow properties of programs. This line of work has been generalized to a probabilistic setting by CertiCrypt [3], which formalizes an extension of RHL for probabilistic programs. CertiPriv builds upon and significantly extends CertiCrypt [3]. The most outstanding difference between the two frameworks is that CertiPriv supports reasoning about a wide range of quantitative relational properties, whereas CertiCrypt is confined to baseline information flow properties. Although we make a modest use of this feature, CertiPriv supports (for a richer language) the certified program transformations that are implemented in CertiCrypt. Thanks to a recent development, the construction of game-playing proofs [6] in CertiCrypt can be achieved efficiently using EasyCrypt [4], a front-end that generates automatically probabilistic RHL derivations using SMT solvers and a verification condition generator. There are exciting opportunities to exploit the synergies between CertiPriv, CertiCrypt and EasyCrypt, as further discussed in Section 8.

There is also a growing body of work that uses proof assistants for reasoning about properties of probabilistic algorithms. For instance, Hurd and co-workers [19] formalized in the HOL system a theory for reasoning about a probabilistic extension of Dijkstra's guarded command language, and used it to prove the correctness of the Miller-Rabin primality test.

8. Future Work and Conclusions

CertiPriv is a machine-checked framework that supports fine-grained reasoning about an expressive class of privacy policies in the Coq proof assistant. In contrast to previous language-based approaches to differential privacy, CertiPriv allows to reason directly about probabilistic computations and to build proofs from first principles. As a result, CertiPriv achieves flexibility, expressiveness, and reliability, and appears as a plausible starting point for capturing and analyzing formally new developments in the field of differential privacy.

An immediate objective for future work is to use the game-playing technique from [3] for verifying in CertiPriv the privacy of multi-party computation algorithms, where one is concerned with ensuring privacy against (computationally bounded) adversaries that only have a partial view of the state, concretely the local state of corrupt participants [5, 25]. This objective is within reach, since CertiPriv inherits from CertiCrypt a formalization of probabilistic polynomial-time algorithms, and can already capture this variant of differential privacy.

Another exciting avenue for further research is to automate the verification of differentially private computations. There are three facets to this work: building an automated checker for apRHL derivations, automatically inferring relational loop invariants, and implementing a precise dependency analysis for an optimal usage of existing composition theorems. EasyCrypt [4] provides an excellent starting point for these tasks.

9. Acknowledgments

This work was partially funded by European Projects FP7-256980 NESSoS and FP7-229599 AMAROUT, Spanish project TIN2009-14599 DESAFIOS 10, Madrid Regional project S2009TIC-1465 PROMETIDOS and French project ANR SESUR-012 SCALP.

References

[1] P. Audebaud and C. Paulin-Mohring. Proofs of randomized algorithms in Coq. *Sci. Comput. Program.*, 74(8):568–589, 2009.

[2] G. Barthe and B. Köpf. Information-theoretic bounds for differentially private mechanisms. In *24rd IEEE Computer Security Foundations Symposium, CSF 2011*, pages 191–204, Los Alamitos, 2011. IEEE Computer Society.

[3] G. Barthe, B. Grégoire, and S. Zanella Béguelin. Formal certification of code-based cryptographic proofs. In *36th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL 2009*, pages 90–101, New York, 2009. ACM.

[4] G. Barthe, B. Grégoire, S. Heraud, and S. Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90, Heidelberg, 2011. Springer.

[5] A. Beimel, K. Nissim, and E. Omri. Distributed private data analysis: Simultaneously solving how and what. In *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 451–468, Heidelberg, 2008. Springer.

[6] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, Heidelberg, 2006. Springer.

[7] N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *31st ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL 2004*, pages 14–25, New York, 2004. ACM.

[8] T.-H. H. Chan, E. Shi, and D. Song. Private and continual release of statistics. In *37th International colloquium on Automata, Languages and Programming, ICALP 2010*, volume 6199 of *Lecture Notes in Computer Science*, pages 405–417, Heidelberg, 2010. Springer.

[9] S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour. Proving programs robust. In *8th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, ESEC/FSE '11*. ACM, 2011.

[10] D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.

[11] M. R. Clarkson and F. B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[12] J. Desharnais, F. Laviolette, and M. Tracol. Approximate analysis of probabilistic processes: Logic, simulation and games. In *5th International Conference on Quantitative Evaluation of Systems, QEST 2008*, pages 264–273. IEEE Computer Society, 2008.

[13] C. Dwork. Differential privacy. In *33rd International Colloquium on Automata, Languages and Programming, ICALP 2006*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12, Heidelberg, 2006. Springer.

[14] C. Dwork. Differential privacy: A survey of results. In *Theory and Applications of Models of Computation*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19, Heidelberg, 2008. Springer.

[15] C. Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54(1):86–95, January 2011.

[16] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503, Heidelberg, 2006. Springer.

[17] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *3rd Theory of Cryptography*

Conference, TCC 2006, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284, Heidelberg, 2006. Springer.

[18] A. Gupta, K. Ligett, F. McSherry, A. Roth, and K. Talwar. Differentially private combinatorial optimization. In *21st Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010*, pages 1106–1125. SIAM, 2010.

[19] J. Hurd, A. McIver, and C. Morgan. Probabilistic guarded commands mechanized in HOL. *Theor. Comput. Sci.*, 346(1):96–112, 2005.

[20] B. Jonsson, W. Yi, and K. G. Larsen. Probabilistic extensions of process algebras. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 685–710. Elsevier, Amsterdam, 2001.

[21] S. P. Kasiviswanathan and A. Smith. A note on differential privacy: Defining resistance to arbitrary side information. Cryptology ePrint Archive, Report 2008/144, 2008.

[22] D. Kifer and A. Machanavajjhala. No free lunch in data privacy. In *2011 International conference on Management of Data, SIGMOD '11*, pages 193–204. ACM Press, 2011.

[23] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2007*, pages 94–103, Washington, 2007. IEEE Computer Society.

[24] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *35th SIGMOD international conference on Management of Data, SIGMOD 2009*, pages 19–30, New York, 2009. ACM.

[25] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan. Computational differential privacy. In *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 126–142, Heidelberg, 2009. Springer.

[26] A. D. Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. *Journal of Computer Security*, 12(1):37–82, 2004.

[27] L. Pitt. A simple probabilistic approximation algorithm for vertex cover. Technical Report TR-404, Yale University, 1985.

[28] N. Ramsey and A. Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *29th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, POPL 2002*, pages 154–165, New York, 2002. ACM.

[29] J. Reed and B. C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In *15th ACM SIGPLAN international conference on Functional programming, ICFP 2010*, pages 157–168, New York, 2010. ACM.

[30] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: security and privacy for MapReduce. In *7th USENIX conference on Networked Systems Design and Implementation, NSDI 2010*, pages 297–312, Berkeley, 2010. USENIX Association.

[31] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *13th IEEE workshop on Computer Security Foundations, CSFW 2000*, pages 200–215, Los Alamitos, 2000. IEEE Computer Society.

[32] R. Segala and A. Turrini. Approximated computationally bounded simulation relations for probabilistic automata. In *20th IEEE Computer Security Foundations symposium, CSF 2007*, pages 140–156, 2007.

[33] T. Terauchi and A. Aiken. Secure information flow as a safety problem. In *12th International Symposium on Static Analysis, SAS 2005*, volume 3672 of *Lecture Notes in Computer Science*, pages 352–367, Heidelberg, 2005. Springer.

[34] The Coq development team. The Coq Proof Assistant Reference Manual Version 8.3. Online – `http://coq.inria.fr`, 2010.

[35] M. C. Tschantz, D. Kaynar, and A. Datta. Formal verification of differential privacy for interactive systems. *Electronic Notes in Theoretical Computer Science*, 276:61–79, 2011.

A. Asymmetric Logic

Asymmetric versions of apRHL can be obtained by re-defining α-distance as

$$\Delta_\alpha(a,b) \overset{\text{def}}{=} \max(b - \alpha a, 0)$$

and dropping in Definition 3 either of the inequalities

$$\begin{array}{c}\Delta_\alpha(\pi_1\,\mu,\mu_1) \leq \delta \\ \Delta_\alpha(\pi_2\,\mu,\mu_2) \leq \delta\end{array}$$

Dropping the second inequality yields a logic for which the validity of a judgment

$$c_1 \sim_{\alpha,\delta} c_2 : \Psi \Rightarrow \Phi$$

implies only that for m_1, m_2 s.t. $m_1 \Psi m_2$ and f_1, f_2 s.t. $f_1 \leq_\Phi f_2$,

$$[\![c_1]\!]\ m_1\ f_1 \leq \alpha([\![c_2]\!]\ m_2\ f_2) + \delta$$

Application to the Minimum Vertex Cover Problem An asymmetric version of the logic would allow to prove in an independent way that for any permutation $\vec{v}$, $\exp(\epsilon)$ is a bound for both, the ratio

$$\frac{\Pr\left[\textsc{VertexCover}(G_1,\epsilon) : \pi = \vec{v}\right]}{\Pr\left[\textsc{VertexCover}(G_2,\epsilon) : \pi = \vec{v}\right]} \tag{1}$$

and its reciprocal. Each ratio could be bounded by applying an asymmetric version of the rule for while loops shown in Figure 5, and each application would in turn require to independently bound for each iteration the ratios

$$\frac{\mathsf{Pr}[v\langle 2\rangle = x]}{\mathsf{Pr}[v\langle 1\rangle = x]} \quad \text{and} \quad \frac{\mathsf{Pr}[v\langle 1\rangle = x]}{\mathsf{Pr}[v\langle 2\rangle = x]}$$

This would allow to choose tighter values for the parameters α_1 and α_2 in each case. E.g., when bounding (1), one could take $\alpha_1(i) = \exp(2/(n-i)w_i)$ and $\alpha_2 = 1$, whereas when bounding its reciprocal one could take $\alpha_1(i) = 1$ and $\alpha_2 = \exp(\epsilon/4)$.