Proving differential privacy in Hoare logic

Gilles Barthe*, Marco Gaboardi‡, Emilio Jesús Gallego Arias†, César Kunz*, Justin Hsu†, and Pierre-Yves Strub*

*IMDEA Software Institute, Spain ‡University of Dundee, Scotland †University of Pennsylvania, USA

***Abstract*—*Differential privacy* is a rigorous, worst-case notion of privacy-preserving computation. Informally, a probabilistic program is differentially private if the participation of a single individual in the input database has a limited effect on the program's distribution on outputs. More technically, differential privacy is a quantitative 2-safety property that bounds the distance between the output distributions of a probabilistic program on adjacent inputs. Like many 2-safety properties, differential privacy lies outside the scope of traditional verification techniques. Existing approaches to enforce privacy are based on intricate, non-conventional type systems, or customized relational logics. These approaches are difficult to implement and often cumbersome to use.**

We present an alternative approach that verifies differential privacy by standard, non-relational reasoning on non-probabilistic programs. Our approach is based on transforming a probabilistic program into a non-probabilistic program which simulates two executions of the original program. We prove that if the target program is correct with respect to a Hoare specification, then the original probabilistic program is differentially private. We provide a variety of examples from the differential privacy literature to demonstrate the utility of our approach. Finally, we compare our approach with existing verification techniques for privacy.

I. Introduction

Program verification provides a rich array of techniques and tools for analyzing program properties. However, program verification is largely confined to reasoning about single program executions, or trace properties. In contrast, many security properties—such as non-interference in information flow systems—require reasoning about *multiple* program executions. These *hyperproperties* [16] encompass many standard security analyses, and lie outside the scope of standard verification tools—to date, there is no generally applicable method or tool for verifying hyperproperties. Instead, ad hoc enforcement methods based on type systems, customized program logics, and finite state automata analyses have been applied to specific hyperproperties. While these approaches can be effective, their design and implementation often require significant effort.

A promising alternative is to reduce verification of a hyperproperty of a program c to verification of a standard property of a transformed program $T(c)$. For instance, *self-composition* [7], [17] is a general method for reducing *2-safety* properties of a program c—which reason about two runs of c—to safety properties of the sequential composition $c; c'$, where c' is a renaming of c. Self-composition is a sound and complete method that applies to many programming languages and verification settings, and has been used to verify information flow properties using standard deductive methods like Hoare logic.

A close relative of self-composition is the *synchronized product* construction. Again, this transformation produces a program which emulates two executions of the original program. While self-composition performs the executions in sequence, synchronized products perform the executions in *lockstep*, dramatically simplifying the verification task for certain properties. This transformation is an instance of the more general class of *product transformations*, as studied by Zaks and Pnueli [40], and later by Barthe et al. [4], [5].

While there has been much research on combining product constructions and deductive verification to reason about 2-safety for deterministic programs, this approach remains largely unexplored for *probabilistic* programs. This is not for lack of interesting use-cases—similar to the case for non-probabilistic computations, many security notions of probabilistic computation are naturally 2-safety properties.

Verifying differential privacy: In this paper, we consider on one such property: *differential privacy*, which provides strong guarantees for privacy-preserving probabilistic computation. Formally, a probabilistic program c is (ϵ, δ)-differentially private with respect to $\epsilon > 0$, $\delta \geq 0$, and a relation[1] Φ on the initial memories of c if for every two initial memories m_1 and m_2 related by Φ, and every subset A of output memories,

$$\Pr[c, m_1 : A] \leq \exp(\epsilon) \Pr[c, m_2 : A] + \delta.$$

Here $\Pr[c, m : A]$ denotes the probability of the output memory landing in A according to distribution $[\![c]\!]\ m$, where $[\![c]\!]$ maps an initial memory m to a distribution $[\![c]\!]\ m$ of output memories. Since this definition concerns two runs of the same probabilistic program, differential privacy is a probabilistic 2-safety property.

Differentially private algorithms are typically built from two constructions: *private mechanisms*, which add probabilistic noise to their input, and *composition*, which combines differentially private operations into a single one. This compositional behavior makes differential privacy an attractive target for program verification efforts.

Existing methods for proving differential privacy have been based on type systems, automata analyses, and customized program logics. For instance, Fuzz [34], DFuzz [24] and

[1]We are here taking a generalization of Differential Privacy with respect to an arbitrary relation Φ. The usual definition is obtained by considering an *adjacency* relation between databases.

related systems [23] enforce differential privacy using linear type systems. This approach is expressive enough to type many examples, but it is currently limited to pure differential privacy (where $\delta = 0$), and cannot handle more advanced examples. Alternatively, Tschantz et al. [39] consider a verification technique based on I/O automata; again, this approach is limited to pure differential privacy. Finally, CertiPriv [11] and EasyCrypt [6] use an approximate relational Hoare logic for probabilistic programs to verify differential privacy. This approach is very expressive and can accommodate approximate differential privacy (when $\delta \neq 0$), but relies on a customized and complex logic. Moreover, ad hoc rules for loops are required for many advanced examples. Each of these approaches also requires non-trivial design and implementation effort.

Self-products for differential privacy: To avoid these drawbacks, we investigate a new approach where proving (ϵ, δ)-differential privacy of a program c is reduced to proving a safety property of a transformed program $T(c)$. In view of previous work verifying 2-safety properties, a natural choice for T is some notion of product program. However, the transformed programs would then be probabilistic, and there are few tools for deductive verification of probabilistic programs. Targeting a non-probabilistic language is more appealing in this regard, as there are many established tools for deductive verification of non-probabilistic programs. Since the original program is a probabilistic program, the key part of our approach is to remove the probabilistic behavior in the target.

To define a transformation T targeting non-probabilistic programs, we proceed in two steps. Starting from a probabilistic program c, we first construct the synchronized product of c with itself. Using the synchronized product instead of self-composition is essential for our second step, in which the probabilistic product program is transformed into a non-probabilistic program.

For this step, we rely on the specific features of differential privacy. First, we observe that differential privacy bounds the ratio—hereafter called the *privacy cost*—between the probabilities of producing the same output on two executions on nearby databases. Second, we recall that there are two main tools for building differentially private computations: private mechanisms, and composition. Private mechanisms and composition interact with the privacy cost in different ways; we consider each in turn.

A private mechanism run over two different inputs returns two closely related distributions, at the cost of consuming some privacy budget. The privacy cost depends on the distance between the inputs: as the two inputs become farther apart, the privacy cost also grows. One fundamental insight (due to Barthe et al. [10]) we use is that the property of being "closely related" can be understood as being at distance 0 for a suitable notion of distance on distributions.

Composition takes a set of differentially private operations and returns the sequential composition of the operations, which is also differentially private. By a property of differential privacy, the privacy cost of the composition is upper bounded by the sum of the privacy costs of the individual operations.

We can now build this reasoning into our verification system. First, we apply the synchronized product construction. Then, we replace two corresponding calls to a mechanism with a call to an abstract procedure that returns equal outputs, at the cost of consuming some privacy budget—roughly, being at distance 0 in the probabilistic setting is collapsed to being equal in the non-probabilistic setting. The second step takes advantage of the synchronized product construction: since the two executions are simulated in lockstep, corresponding calls to a mechanism are next to each other in the product program. Since mechanisms are the only probabilistic parts of our source program, our output program is now non-probabilistic.

To keep track of the privacy cost, we use ghost variables $\mathtt{v}_\epsilon$ and $\mathtt{v}_\delta$ which are incremented after each mechanism is executed, in terms of the distance between their two inputs.

To illustrate our approach, consider the following simple program c:

$$s;\ x \leftarrow \mathsf{Lap}_\epsilon(e);\ \mathsf{return}\ x$$

where s is a deterministic computation and Lap is the Laplace mechanism—a probabilistic operator that achieves differential privacy by adding noise to its input. The synchronized product $T(c)$ of the program c is

$$T(s);\ x_1 \leftarrow \mathsf{Lap}_\epsilon(e_1);\ x_2 \leftarrow \mathsf{Lap}_\epsilon(e_2);\ \mathsf{return}\ (x_1, x_2)$$

where $T(s)$ is the synchronized product of s. Then, we make the program non-probabilistic by replacing the two calls to the Laplace mechanism with a call to an abstract procedure $\mathsf{Lap}^\diamond$, giving the following transformed program $T(c)$.

$$T(s); (x_1, x_2) \leftarrow \mathsf{Lap}^\diamond(e_1, e_2); \mathsf{return}\ (x_1, x_2)$$

Roughly, the specification of the procedure invocation $\mathsf{Lap}^\diamond$ states that the same value is assigned to x_1 and x_2. Also, as side effect, the variable $\mathtt{v}_\epsilon$ is updated to increment the privacy cost, which depends on the distance between the inputs (e_1, e_2) to the Laplace mechanism.

Our main result (Theorem 5 in §III) states that once we perform this transformation, we can use plain Hoare logic to complete the verification. More concretely, for the example above, c is $(\epsilon, 0)$-differentially private if the following Hoare specification is valid.

$$\vdash T(c) : \Phi \wedge \mathtt{v}_\epsilon = 0 \Longrightarrow x_1 = x_2 \wedge \mathtt{v}_\epsilon \leq \epsilon_0$$

Contributions: The main contribution of the paper (§III) is a program transformation that operates on programs built from sequential, non-probabilistic constructs and differentially private, probabilistic primitives—such as the Laplace and Exponential mechanisms. The transformed program is non-probabilistic, and the differential privacy of the original program can be reduced to a safety property of the transformed program. Then we show in §IV that our approach subsumes the core apRHL logic of Barthe et al. [11], in the sense that every algorithm provable with core apRHL is also provable with our approach

We illustrate the expressiveness of our approach in §V by verifying differential privacy of several probabilistic algorithms, including a recent algorithm that produces synthetic datasets using a combination of the multiplicative weights update rule and of the exponential mechanism [28], [27], and the Propose-Test-Release (PTR) framework [20], [38], which achieves approximate differential privacy without relying on output perturbation. Finally, we discuss the example of vertex cover, which is provable with an ad hoc extension of core apRHL, but cannot be handled directly by our approach.

II. A PRIMER ON DIFFERENTIAL PRIVACY

Let us begin by recalling the basic definitions of differential privacy.

Definition 1: Let $\epsilon, \delta \geq 0$, and let $\Phi \subseteq \mathcal{S} \times \mathcal{S}$ be a relation on $\mathcal{S}$. A randomized algorithm K taking inputs in $\mathcal{S}$ and returning outputs in $\mathcal{R}$ is (ϵ, δ)-*differentially private with respect to* Φ if for every two inputs $s_1, s_2 \in \mathcal{S}$ such that $s_1 \ \Phi \ s_2$ and every subset of outputs $A \subseteq \mathcal{R}$,

$$\Pr\left[K(s_1) : A\right] \leq \exp(\epsilon) \ \Pr\left[K(s_2) : A\right] + \delta.$$

When $\delta = 0$, we will call this ϵ-*differentially privacy*.

Our definition is a variant of the standard definition [21] where input memories are considered to be databases, and Φ relates databases that differ in a single individual's data. We explain the intuition of differential privacy in this. Recall that differential privacy aims to conceal the participation of individuals in a study. To distinguish between the participation or non-participation of an individual, we say that two databases D and D' are *adjacent* or *neighboring* if they differ only in the presence or absence of a single record; note that the adjacency relation is necessarily symmetric.

Differential privacy then states that the distributions output by K on adjacent databases are close. In the simple case where $\delta = 0$, the definition above requires that the probability of any output changes by at most a $\mathsf{exp}(\epsilon)$ factor when moving from one input to an adjacent input. When $\delta > 0$ these bounds are still valid except with probability δ. In other words, δ is a probability of failure in ensuring a privacy bound.

Building private programs: Let F be a deterministic computation with inputs in $\mathcal{T}$ and outputs in $\mathcal{R}$. Suppose that we want to make the computation of F (ϵ, δ)-differentially private with respect to some relation Φ. A natural way to achieve this goal is to add random noise to the evaluation of F on an input. In general, the noise that we need to add depends not only on the ϵ and δ parameters (which control the strength of the privacy guarantee), but also on the *sensitivity* of F, a quantity that is closely related to Lipschitz continuity for functions.

Definition 2: Assume that F is real-valued, i.e. $\mathcal{R} = \mathbb{R}$, and let $k > 0$. We say that F is k-*sensitive with respect to* Φ if $|F(t_1) - F(t_2)| \leq k$ for all $t_1, t_2 \in \mathcal{T}$ such that $t_1 \ \Phi \ t_2$.

The canonical mechanism for privately releasing a k-sensitive function is the *Laplace mechanism*.

Theorem 1 ([18]): Suppose $\epsilon > 0$. The *Laplace mechanism* is defined by

$$\mathrm{Lap}_\epsilon(t) = t + v,$$

where v is drawn from the Laplace distribution $\mathcal{L}(1/\epsilon)$, i.e. with probability density function

$$P(v) = \mathsf{exp}(-\epsilon|v|).$$

If F is k-sensitive with respect to Φ, then the probabilistic function that maps t to $\mathrm{Lap}_\epsilon(F(t))$ is $(k\epsilon, 0)$-differentially private with respect to Φ.

Additionally, the Laplace mechanism satisfies a simple accuracy bound.

Lemma 1: Let $\epsilon, \delta > 0$ and let $T = \log(2/\delta)/(2\epsilon)$. Then for every x, $\mathsf{Lap}_\epsilon(x) \in (x - T, x + T)$ with probability at least $1 - \delta$.

Another mechanism that is fundamental for differential privacy is the *Exponential mechanism* [32]. Let $\mathcal{T}$ be the set of inputs, typically thought of as the private information. Let $\mathcal{R}$ be a set of outputs, and consider a function $F : \mathcal{T} \times \mathcal{R} \to \mathbb{R}$, typically called the *score function*. We first extend the definition of sensitivity to this function.

Definition 3: Assume $F : \mathcal{T} \times \mathcal{R} \to \mathbb{R}$ and let $c > 0$. We say that F is k-*sensitive on* $\mathcal{T}$ *with respect to* Φ if $|F(t_1, r) - F(t_2, r)| \leq k$ for all $t_1, t_2 \in \mathcal{T}$ such that $t_1 \ \Phi \ t_2$ and $r \in \mathcal{R}$.

Then, the Exponential mechanism can be used to output an element of $\mathcal{R}$ that approximately maximizes the score function, if the score function is k-sensitive.

Theorem 2 ([32]): Let $\epsilon, c > 0$. Suppose that F is k-sensitive in $\mathcal{T}$ with respect to Φ. The *Exponential mechanism*[2] $\mathrm{Exp}_\epsilon(F, t)$ takes as input $t \in \mathcal{T}$, and returns $r \in \mathcal{R}$ with probability equal to

$$\frac{\mathsf{exp}(\epsilon F(t, r)/2)}{\sum_{r' \in \mathcal{R}} \mathsf{exp}(\epsilon F(t, r')/2)}.$$

This mechanism is $(k\epsilon, 0)$-differentially private with respect to Φ.

A powerful feature of differential privacy is that by composing differentially private mechanisms, we can construct new mechanisms that satisfy differential privacy. However, the privacy guarantee will degrade: more operations on a database will lead to more privacy loss. This is formalized by the following composition theorem.

Theorem 3 ([31]): Let q_1 be a (ϵ_1, δ_1)-differentially private query and let q_2 be a (ϵ_2, δ_2)-differentially private query. Then, their composition $q(t) = (q_1(t), q_2(t))$ is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-differentially private.

In light of this composition property, we will often think of the privacy parameters ϵ and δ of a program as *privacy budgets* that are consumed by sub-operations. Finally, differential privacy is closed under *post-processing*—an output of a private algorithm can be arbitrarily transformed, so long as this processing does not involve the private database.

Theorem 4: Let q be (ϵ, δ)-differentially private mapping databases to some output range R, and let $f : R \to R'$ be an arbitrary function. Then, the post-processing $f \circ q$ is also (ϵ, δ)-differentially private.

[2]The Exponential mechanism as first introduced by McSherry and Talwar [32] is parameterized by a prior distribution μ on $\mathcal{R}$. We consider the special case where μ is uniform; this suffices for typical applications.

III. Self-products

In this section, we formalize the verification of differential-privacy using traditional Hoare logic. We start with some preliminary definitions and the pWhile programming language, which will serve as our source language. Then, given a *probabilistic* pWhile program c, we show how to build a *non-probabilistic* program $T(c)$ that simulates two executions of c on different inputs and tracks the privacy cost via two ghost variables $\mathtt{v}_\epsilon$ and $\mathtt{v}_\delta$. We show that the verification of $T(c)$ with respect to a Hoare logic specification ensures differential privacy of the original program c.

A. Distributions

We define the set $\mathcal{D}(A)$ of *sub-distributions* over a set A as the set of functions $\mu : A \to [0,1]$ with discrete $\mathsf{support}(\mu) = \{x \mid \mu\, x \neq 0\}$, such that $\sum_{x \in A} \mu\, x \leq 1$; when equality holds, μ is a true *distribution*. (We will often refer to sub-distributions as distributions when there is no confusion.) Sub-distributions can be given the structure of a complete partial order: for all $\mu_1, \mu_2 \in \mathcal{D}(A)$,

$$\mu_1 \sqsubseteq \mu_2 \ \stackrel{\text{def}}{=}\ \forall a \in A.\ \mu_1\, a \leq \mu_2\, a.$$

Moreover, sub-distributions can be given the structure of a monad: for any function $g : A \to \mathcal{D}(B)$ and distribution $\mu : \mathcal{D}(A)$, we define $g^\star\, \mu : \mathcal{D}(B)$ to be the following sub-distribution:

$$g^\star\, \mu\, (b) \ \stackrel{\text{def}}{=}\ \sum_{a \in A} (g\, a\, b)(\mu\, a),$$

for every $b \in B$. Given an element $a \in A$ we denote by $\mathbb{1}_a$ the probability distribution returning a with probability one.

In the following we will use a normalization construction $(\cdot)^\#$ that takes as input a function $f : B \to \mathbb{R}^{\geq 0}$ over a discrete set B and returns $(f)^\# : \mathcal{D}(B)$ such that the probability mass of $f^\#$ at b is given by

$$(f)^\#\, b \ \stackrel{\text{def}}{=}\ \frac{f\, b}{\sum_{b' \in B} f\, b'}.$$

Intuitively, sampling from the distribution $(f)^\#$ is equivalent to sampling "with probability proportional to" f.

B. pWhile *Language*

pWhile programs will serve as our source language, and are defined by the following grammar:

$$\begin{array}{rll}
\mathcal{C} ::= & \mathsf{skip} & \\
\mid & \mathcal{C}; \mathcal{C} & \text{sequencing} \\
\mid & \mathcal{V} \leftarrow \mathcal{E} & \text{deterministic assignment} \\
\mid & \mathcal{V} \xleftarrow{\$} \mathsf{Lap}_\epsilon(\mathcal{E}) & \text{Laplace assignment} \\
\mid & \mathcal{V} \xleftarrow{\$} \mathsf{Exp}_\epsilon(\mathcal{E}, \mathcal{E}) & \text{Exponential assignment} \\
\mid & \mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C} & \text{conditional} \\
\mid & \mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C} & \text{while loop} \\
\mid & \mathsf{return}\ \mathcal{E} & \text{return expression}
\end{array}$$

Here, $\mathcal{V}$ is a set of *variables* and $\mathcal{E}$ is a set of *expressions*. We consider expressions including simply typed lambda terms and basic operations on booleans, lists and integers. The probabilistic assignments involving $\mathsf{Lap}_\epsilon(\mathcal{E})$ and $\mathsf{Exp}_\epsilon(\mathcal{E}, \mathcal{E})$ internalize the (discrete version of the) mechanisms of Theorem 1 and Theorem 2 respectively. For simplicity, we only consider commands of the form $c; \mathsf{return}\ e$ in the rest of this paper. pWhile is equipped with a standard type system; we omit the typing rules. Note that for examples based on the exponential mechanism we allow function types for representing the score functions; alternatively these score functions can be modeled as finite maps if their domain is finite (as will be the case in our examples).

The semantics of a well-typed pWhile program is defined by its (probabilistic) action on *memories*; we denote the set of memories by $\mathcal{M}$. A program memory $m \in \mathcal{M}$ is a partial assignment of values to variables. Formally, the semantics of a return-free pWhile program c is a function $[\![c]\!] : \mathcal{M} \to \mathcal{D}(\mathcal{M})$ mapping a memory $m \in \mathcal{M}$ to a distribution $[\![c]\!]\, m \in \mathcal{D}(\mathcal{M})$, as defined in Fig. 1. Then, the semantics of a program $c; \mathsf{return}\ e$ is simply defined as

$$[\![c;\ \mathsf{return}\ e]\!]\ m \ \stackrel{\text{def}}{=}\ \mathbb{1}^\star_{[\![e]\!]}\, ([\![c]\!]\, m).$$

C. Target Language

To define the target language of our transformation, we remove probabilistic assignments, and add assert and havoc instructions to pWhile, giving the following grammar:

$$\begin{array}{rll}
\mathcal{C} ::= & \mathsf{skip} & \\
\mid & \mathcal{C}; \mathcal{C} & \text{sequencing} \\
\mid & \mathcal{V} \leftarrow \mathcal{E} & \text{deterministic assignment} \\
\mid & \mathsf{assert}\ (\varphi) & \text{assert} \\
\mid & (\mathcal{V}, \mathcal{V}) \leftarrow \mathsf{Lap}^\diamond_\epsilon(\mathcal{E}, \mathcal{E}) & \text{Laplace invocation} \\
\mid & (\mathcal{V}, \mathcal{V}) \leftarrow \mathsf{Exp}^\diamond_\epsilon(\mathcal{E}, \mathcal{E}, \mathcal{E}, \mathcal{E}) & \text{Exponential invocation} \\
\mid & \mathsf{if}\ \mathcal{E}\ \mathsf{then}\ \mathcal{C}\ \mathsf{else}\ \mathcal{C} & \text{conditional} \\
\mid & \mathsf{while}\ \mathcal{E}\ \mathsf{do}\ \mathcal{C} & \text{while loop} \\
\mid & \mathsf{return}\ \mathcal{E} & \text{return expression}
\end{array}$$

The semantics of most of the constructions of this language is standard, except for a few parts. The $\mathsf{assert}\ (\varphi)$ statement checks at runtime whether the predicate φ is valid, and stops the execution otherwise. We defer the presentation of the abstract procedures $\mathsf{Lap}^\diamond$ and $\mathsf{Exp}^\diamond$ until the definition of the self-product construction, in §III-E.

The enforcement of safety properties over this target language is formalized by a standard Hoare logic, with judgments of the form

$$\vdash c : \Psi \Longrightarrow \Phi.$$

Here the pre- and post-conditions Ψ and Φ are standard *unary* predicates over memories. Hoare logic judgments can be derived using the rules in Fig. 2; by the standard soundness of Hoare logic, the derivability of a judgment $\vdash c : \Psi \Longrightarrow \Phi$ entails the correctness of c with respect to its specification Ψ, Φ.

D. Product Construction

Before we define the product transformation from pWhile to our target language, let us first review some preliminaries about product programs.

$$
\begin{array}{lcl}
[\![\mathsf{skip}]\!]\, m & = & \mathbb{1}_m \\
[\![c_1;\, c_2]\!]\, m & = & [\![c_2]\!]^\star\, ([\![c_1]\!]\, m) \\
[\![x \leftarrow e]\!]\, m & = & \mathbb{1}_{m\{[\![e]\!]_{\mathcal{E}}\, m/x\}} \\
[\![x \xleftarrow{\$} \mathsf{Lap}_\epsilon(e)]\!]\, m & = & \left(\lambda v.\ \mathbb{1}_{m\{v/x\}}\right)^\star \left(\lambda r. \mathsf{exp}\left(-\frac{\epsilon|r - [\![e]\!]\, m|}{2}\right)\right)^\# \\
[\![x \xleftarrow{\$} \mathsf{Exp}_\epsilon(s, e)]\!]\, m & = & \left(\lambda v.\ \mathbb{1}_{m\{v/x\}}\right)^\star \left(\lambda r. \mathsf{exp}\left(\frac{\epsilon[\![s]\!] m([\![e]\!] m, r)}{2}\right)\right)^\# \\
[\![\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!]\, m & = & \mathbf{if}\ ([\![e]\!]_{\mathcal{E}}\, m = \mathsf{true})\ \mathbf{then}\ ([\![c_1]\!]\, m)\ \mathbf{else}\ ([\![c_2]\!]\, m) \\
[\![\mathsf{while}\ e\ \mathsf{do}\ c]\!]\, m & = & \bigsqcup w_i\, m \\
\quad \text{where} \quad w_0\, m & = & \bot \\
\quad\quad\quad w_{i+1}\, m & = & \mathbf{if}\ ([\![e]\!]_{\mathcal{E}}\, m = \mathsf{true})\ \mathbf{then}\ w_i^\star\, ([\![c]\!]\, m)\ \mathbf{else}\ \mathsf{unit}\, m
\end{array}
$$

Fig. 1: pWHILE semantics

$$
\frac{}{\vdash \mathsf{skip} : \Psi \Longrightarrow \Psi} \qquad \frac{}{\vdash x \leftarrow e : \Phi\, \{e/x\} \Longrightarrow \Phi} \qquad \frac{}{\vdash \mathsf{assert}\, (\varphi) : \Phi \wedge \varphi \Longrightarrow \Phi}
$$

$$
\frac{\vdash c_1 : \Psi \Longrightarrow \varphi \qquad \vdash c_2 : \varphi \Longrightarrow \Phi}{\vdash c_1;\, c_2 : \Psi \Longrightarrow \Phi} \qquad \frac{\vdash c_1 : \Psi \wedge b \Longrightarrow \Phi \qquad \vdash c_2 : \Psi \wedge \neg b \Longrightarrow \Phi}{\vdash \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 : \Psi \Longrightarrow \Phi}
$$

$$
\frac{\Psi \wedge v \leq 0 \Rightarrow \neg b \qquad \vdash c : \Psi \wedge b \wedge v = k \Longrightarrow \Psi \wedge v < k}{\vdash \mathsf{while}\ b\ \mathsf{do}\ c : \Psi \Longrightarrow \Psi \wedge \neg b} \qquad \frac{\vdash c : \Psi' \Longrightarrow \Phi' \qquad \Psi \Rightarrow \Psi' \qquad \Phi' \Rightarrow \Phi}{\vdash c : \Psi \Longrightarrow \Phi}
$$

Fig. 2: Hoare logic for non-probabilistic programs

Product programs have been successfully used to verify 2-safety properties like information-flow, program equivalence, and program robustness. As mentioned above, a *synchronized* product program can be used to simulate two runs of the same program, interleaving the two executions and often simplifying the verification effort. This technique, however, has been mostly used in the verification of non-probabilistic programs. In the rest of this section we provide a brief introduction to relational verification by product construction and then extend this approach to handle quantitative reasoning over probabilistic programs.

A simple but necessary concept for the product construction is *memory separability*: we say that two programs are *separable* if they manipulate disjoint sets of program variables. In order to achieve separability in the construction of the product of a program with itself, program variables are renamed with a left ($-_1$) or right ($-_2$) tag. For any program expression e or predicate φ, we let e_i and φ_i stand for the result of renaming every program variable with the tag $-_i$.

Similarly, we say that two memories are *disjoint* when their domains (the sets of variables on which they are defined) are disjoint. Notice that given two disjoint memories m_1 and m_2, we can build a memory $m = m_1 \oplus m_2$ representing their union. In the following, we exploit separability and use predicates to represent *binary relations* over disjoint memories m_1 and m_2. We will suggestively write $m_1\, \Phi\, m_2$ to denote the unary predicate $\Phi(m_1 \oplus m_2)$ over the combined memory $m_1 \oplus m_2$.

Given two deterministic programs c_1 and c_2, a general *product program* $c_1 \times c_2$ is a syntactic construction that merges the executions of c_1 and c_2; this construction is required to correctly represent every pair of executions of c_1 and c_2. Traditional program verification techniques can then be used to enforce a relational property over c_1 and c_2.

In self-composition [7], [17], the product construction $c_1 \times c_2$ is defined simply by the sequential composition $c_1;\, c_2$. An inconvenience of self-composition is that the verification of $c_1;\, c_2$ usually requires independent functional reasoning over c_1 and c_2. The synchronized product construction solves this problem by *interleaving* execution of two runs of the same program—by placing corresponding pieces of the two executions of a program close together, synchronized product programs can more easily maintain inductive invariants relating the two runs. Not only does synchronization reduce the verification effort, we will soon see that synchronization is the key feature that makes our verification approach possible.

E. Building the Product

We embed the quantitative reasoning on probabilistic programs by introducing the special program variables v_ϵ and v_δ, which serve to accumulate the privacy cost. For every statement c, the self-product $\lceil c \rceil$ is formally defined by the rules shown in Fig. 4. In a nutshell, the deterministic fragment of the code is duplicated with appropriate variable renaming with the flags $-_1$ and $-_2$, and the control flow is *fully synchronized*, i.e., the two executions of the same program must take all the same branches. We use the assert statements

$$
\begin{array}{lcl}
\lceil \mathsf{skip} \rceil & = & \mathsf{skip} \\
\lceil c_1;\, c_2 \rceil & = & \lceil c_1 \rceil;\, \lceil c_2 \rceil \\
\lceil x \leftarrow e \rceil & = & x_1 \leftarrow e_1;\, x_2 \leftarrow e_2 \\
\lceil x \xleftarrow{\$} \mathsf{Lap}_\epsilon(e) \rceil & = & (x_1, x_2) \leftarrow \mathsf{Lap}^\diamond(e_1, e_2) \\
\lceil x \xleftarrow{\$} \mathsf{Exp}_\epsilon(s, e) \rceil & = & (x_1, x_2) \leftarrow \mathsf{Exp}^\diamond(s_1, e_1, s_2, e_2) \\
\lceil \mathsf{if}\ b\ \mathsf{then}\ c\ \mathsf{else}\ d \rceil & = & \mathsf{assert}\,(b_1 = b_2); \\
 & & \mathsf{if}\ b_1\ \mathsf{then}\ \lceil c \rceil\ \mathsf{else}\ \lceil d \rceil \\
\lceil \mathsf{while}\ b\ \mathsf{do}\ c \rceil & = & \mathsf{assert}\,(b_1 = b_2); \\
 & & \mathsf{while}\ b_1\ \mathsf{do} \\
 & & \quad \lceil c \rceil;\, \mathsf{assert}\,(b_1 = b_2)
\end{array}
$$

Fig. 4: Self-product construction

to enforce this property. Moreover, for the self-product of a program c to correctly represent two executions of itself, we require that loop guards do not depend on probabilistically sampled values; we assume in the remainder of this work that the programs under verification satisfy this condition. Similarly, we assume that programs under verification contain only terminating while loops.

The probabilistic constructions are mapped to invocations to the abstract procedures $\mathsf{Lap}^\diamond$ and $\mathsf{Exp}^\diamond$. The semantics of these procedures is non-deterministic, in order to simulate an assignment for every value that can be sampled from the probability distribution. We axiomatize these abstract procedures with Hoare specifications. Figure 3 extends the set of Hoare logic rules with a specification for the invocation of $\mathsf{Lap}^\diamond$ and $\mathsf{Exp}^\diamond$. Notice that both abstract procedures have an incremental side effect over the privacy budget variable $\mathtt{v}_\epsilon$. In Section V-C, we introduce a modified specification for $\mathsf{Lap}^\diamond$ that also increments the budget variable $\mathtt{v}_\delta$.

F. An alternative characterization of privacy

For the proof of soundness, we will use an alternative characterization of (ϵ, δ)-differential privacy based on the notion of ϵ-distance. This notion is adapted from the asymmetric notion of distance used by Barthe et al. [11].

Definition 4 (ϵ-distance): The ϵ-distance Δ_ϵ is defined as

$$\Delta_\epsilon(\mu_1, \mu_2) \stackrel{\text{def}}{=} \max_{S \subseteq A} (\mu_1\, S - \exp(\epsilon)\, \mu_2\, S),$$

where $\mu\, S \stackrel{\text{def}}{=} \sum_{a \in S} \mu\, a$. Note that we define max over an empty set to be 0, so $\Delta_\epsilon(\mu_1, \mu_2) \geq 0$.

By the definition of ϵ-distance, a probabilistic program c is (ϵ, δ)-differentially private with respect to $\epsilon > 0$, $\delta \geq 0$, and a relation Φ on the initial memories of c if for every two memories m_1 and m_2 related by Φ, we have

$$\Delta_\epsilon(\llbracket c \rrbracket\ m_1, \llbracket c \rrbracket\ m_2) \leq \delta.$$

The proof of our main theorem relies on a *lifting* operator that turns a relation on memories into a relation on *distributions* over memory. Given a relation on memories Φ, and real values ϵ, δ we define the lifted relation on memory distributions $\Phi_{\langle \epsilon, \delta \rangle}$ as follows.

Definition 5: For all memory distributions μ_1, μ_2, $\mu_1\, \Phi_{\langle \epsilon, \delta \rangle}\, \mu_2$ if there exists μ such that:

1) $\pi_i\, \mu \leq \mu_i$,
2) $\forall m, \mu\, m \neq 0 \Rightarrow \Phi\, m$, and
3) $\Delta_\epsilon(\mu_i, \pi_i\, \mu) \leq \delta$,

where

- $(\pi_1\, \mu)\, m_1 = \sum_{m_2 \in \mathcal{M}} \mu\, (m_1, m_2)$, and
- $(\pi_2\, \mu)\, m_2 = \sum_{m_1 \in \mathcal{M}} \mu\, (m_1, m_2)$.

Notice that ϵ-distance between distributions is closely related to the lifting of the equality relation, i.e.,

$$\mu_1 =_{\langle \epsilon, \delta \rangle} \mu_2 \iff \Delta_\epsilon(\mu_1, \mu_2) \leq \delta. \tag{1}$$

Note that the second equation is precisely the condition on output distributions needed for (ϵ, δ)-differential privacy.

G. Soundness of the self-product technique

The following result states the soundness of our approach.

Theorem 5: If the following Hoare judgment is valid

$$\vdash \lceil c \rceil : \Psi \wedge \mathtt{v}_\epsilon = 0 \wedge \mathtt{v}_\delta = 0 \Longrightarrow \mathtt{out}_1 = \mathtt{out}_2 \wedge \mathtt{v}_\epsilon \leq \epsilon \wedge \mathtt{v}_\delta \leq \delta$$

then c satisfies (ϵ, δ)-differential privacy.

Proof: We prove a generalization. Let Φ be a relation on memories, and suppose

$$\vdash \lceil c \rceil : \Psi \wedge \mathtt{v}_\epsilon = 0 \wedge \mathtt{v}_\delta = 0 \Longrightarrow \Phi \wedge \mathtt{v}_\epsilon \leq \epsilon \wedge \mathtt{v}_\delta \leq \delta.$$

Then, for all memories m_1, m_2 such that $m_1\, \Psi\, m_2$, we have

$$(\llbracket c \rrbracket\, m_1)\, \Phi_{\langle \epsilon, \delta \rangle}\, (\llbracket c \rrbracket\, m_2).$$

The proof follows by structural induction on c; we provide technical details in the appendix. ∎

IV. Comparison with apRHL

Now that we have defined our transformation, we compare our approach to a custom logic for verifying privacy. apRHL [11] is a quantitative, probabilistic and relational program logic for reasoning about differential privacy, with judgments of the form[3]

$$\vdash c_1 \sim_{\langle \alpha, \delta \rangle} c_2 : \Psi \Longrightarrow \Phi,$$

where c_1 and c_2 are probabilistic programs, Ψ and Φ are memory relations, and ϵ, δ are real values. The main result of apRHL states that if $\vdash c_1 \sim_{\langle \epsilon, \delta \rangle} c_2 : \Psi \Longrightarrow \mathtt{out}_1 = \mathtt{out}_2$ is derivable, where c_1 and c_2 are the result of renaming variables in c to make them separable, then c is (ϵ, δ)-differentially private with respect to the relation Ψ on initial memories.

Fig. 5 shows an excerpt of the core rules of the apRHL logic, including rules for the Laplace and Exponential mechanisms. Our approach subsumes core apRHL; the following lemma shows that every probabilistic program c that can be verified (ϵ, δ)-differentially private using the apRHL logic rules shown in Fig. 5 can be verified using our self-product technique. The soundness of the sequential composition rule

[3]The original apRHL rules are based on a multiplicative privacy budget. We adapt the rules to an additive privacy parameter for consistency with the rest of the article.

$$\frac{}{\vdash (x_1, x_2) \leftarrow \mathsf{Lap}^{\diamond}_{\epsilon}(e_1, e_2) : \mathrm{v}_\epsilon = \epsilon_0 \wedge \mathrm{v}_\delta = \delta_0 \Longrightarrow x_1 = x_2 \wedge \mathrm{v}_\epsilon = \epsilon_0 + |e_1 - e_2|\epsilon \wedge \mathrm{v}_\delta = \delta_0}$$

$$\frac{}{\vdash (x_1, x_2) \leftarrow \mathsf{Exp}^{\diamond}_{\epsilon}(s_1, e_1, s_2, e_2) : s_1 = s_2 \wedge \mathrm{v}_\epsilon = \epsilon_0 \wedge \mathrm{v}_\delta = \delta_0 \Longrightarrow x_1 = x_2 \wedge \mathrm{v}_\epsilon = \epsilon_0 + \epsilon \max_r |s_1(x_1, r) - s_2(x_2, r)|}$$

Fig. 3: Hoare specification for $\mathsf{Lap}^{\diamond}$ and $\mathsf{Exp}^{\diamond}$

$$\frac{}{\vdash x_1 \leftarrow e_1 \sim_{\langle 0,0\rangle} x_2 \leftarrow e_2 : \Phi\,\{e_1/x_1\}\,\{e_2/x_2\} \Longrightarrow \Phi}\,[\mathsf{assn}]$$

$$\frac{}{\vdash y_1 \overset{\$}{\leftarrow} \mathsf{Lap}_\epsilon(e_1) \sim_{\langle |e_1-e_2|\epsilon, 0\rangle} y_2 \overset{\$}{\leftarrow} \mathsf{Lap}_\epsilon(e_2) : \mathsf{true} \Longrightarrow y_1 = y_2}\,[\mathsf{lap}]$$

$$\frac{}{\vdash y_1 \overset{\$}{\leftarrow} \mathsf{Exp}_\epsilon(s_1, e_1) \sim_{\langle \epsilon \max_r |s_1(x_1,r)-s_2(x_2,r)|, 0\rangle} y_2 \overset{\$}{\leftarrow} \mathsf{Exp}_{\epsilon,s}(s_2, e_2) : s_1 = s_2 \Longrightarrow y_1 = y_2}\,[\mathsf{exp}]$$

$$\frac{}{\vdash \mathsf{skip} \sim_{\langle 0,0\rangle} \mathsf{skip} : \Psi \Longrightarrow \Psi}\,[\mathsf{skip}]$$

$$\frac{\vdash c_1 \sim_{\langle \epsilon,\delta\rangle} c_2 : \Psi \wedge b_1 \Longrightarrow \Phi \qquad \vdash d_1 \sim_{\langle \epsilon,\delta\rangle} d_2 : \Psi \wedge \neg b_1 \Longrightarrow \Phi}{\vdash \mathsf{if}\ b_1\ \mathsf{then}\ c_1\ \mathsf{else}\ d_1 \sim_{\langle \epsilon,\delta\rangle} \mathsf{if}\ b_2\ \mathsf{then}\ c_2\ \mathsf{else}\ d_2 : \Psi \wedge b_1 = b_2 \Longrightarrow \Phi}\,[\mathsf{cond}]$$

$$\frac{\begin{array}{c}\vdash c_1 \sim_{\langle \epsilon,\delta\rangle} c_2 : \Theta \wedge b_1 \wedge k = e \Longrightarrow \Theta \wedge k < e \\ \Theta \wedge n \le e \implies \neg b_1 \qquad \Theta \implies b_1 = b_2\end{array}}{\vdash \mathsf{while}\ b_1\ \mathsf{do}\ c_1 \sim_{\langle n\epsilon, n\delta\rangle} \mathsf{while}\ b_2\ \mathsf{do}\ c_2 : \Theta \wedge 0 \le e \Longrightarrow \Theta \wedge \neg b_1}\,[\mathsf{while}]$$

$$\frac{\vdash c_1 \sim_{\langle \epsilon,\delta\rangle} c_2 : \Psi \Longrightarrow \Phi' \qquad \vdash c_1' \sim_{\langle \epsilon',\delta'\rangle} c_2' : \Phi' \Longrightarrow \Phi}{\vdash c_1; c_1' \sim_{\langle \epsilon+\epsilon', \delta+\delta'\rangle} c_2; c_2' : \Psi \Longrightarrow \Phi}\,[\mathsf{seq}]$$

$$\frac{\vdash c_1 \sim_{\langle \epsilon',\delta'\rangle} c_2 : \Psi' \Longrightarrow \Phi' \qquad \Psi \Rightarrow \Psi' \qquad \Phi' \Rightarrow \Phi \qquad \epsilon' \le \epsilon \qquad \delta' \le \delta}{\vdash c_1 \sim_{\langle \epsilon,\delta\rangle} c_2 : \Psi \Longrightarrow \Phi}\,[\mathsf{weak}]$$

Fig. 5: Core proof rules of the approximate relational Hoare logic

[seq] relies on the assumption that while loops are terminating.

Lemma 2: For every probabilistic program c, memory relations Ψ, Φ and real expressions ϵ, δ such that the following apRHL judgment is derivable

$$\vdash c \sim_{\langle \epsilon,\delta\rangle} c : \Psi \Longrightarrow \Phi$$

we have

$$\vdash \lceil c \rceil : \Psi \Longrightarrow \Phi \wedge \mathrm{v}_\epsilon \le \epsilon \wedge \mathrm{v}_\delta \le \delta.$$

The proof of this result is straightforward, by induction on the derivation of the apRHL judgement.

Furthermore, our system can verify examples not captured by core apRHL—for example, the *smart sum* algorithm we describe in §V-A. Our approach is more expressive because privacy consumption in apRHL is tracked by an accumulator which is part of the judgment itself, independent of the pre-condition and the initial memory. Using self-products, reasoning about the privacy budget is carried out in the Hoare specification and consequently inherits the full expressivity of the Hoare logic.

Another instance of an algorithm that cannot be verified in core apRHL is the *minimum vertex cover* algorithm developed by Gupta et al. [25]. The algorithm can proved differentially private in an ad hoc extension of apRHL. One can extend self-products to consider the minimum vertex cover algorithm, and prove the pre-condition of Theorem 5; however, extending the proof of Theorem 5 to account for this example is problematic. §V-D discusses this example in more detail.

V. Examples

In this section, we apply our method to four examples. The first example (smart sum) is an algorithm for computing statistics; it involves intricate applications of the composition theorem, and is thus an interesting test case. The second example (Iterative Database Construction, or more precisely the Multiplicative Weights Exponential Mechanism) is an algorithm that computes a synthetic database; it combines the Laplace and the Exponential mechanisms, and has not been verified in earlier work using relational logic. The third example (Propose-Test-Release) is an algorithm that only achieves approximate differential privacy (i.e., (ϵ, δ)-differential privacy with $\delta > 0$) using both the privacy and accuracy properties of the Laplace distribution. To best of our knowledge, we provide the first machine-checked proof of this mechanism. Finally, our last example (vertex cover) is an algorithm that achieves differential privacy by carefully adding noise to sampled values; this example can only be verified partially using our method, and illustrates the differences with apRHL.

A. Smart sum

In this example, a database db is a list of real numbers $[r_1, \ldots, r_T]$ and we consider two databases *adjacent* if they are the same length T, at most one entry differs between the two databases, and that entry differs by at most 1.

Suppose we want to release private sums of the first i entries, simultaneously for every $i \in [1 \ldots T]$: that is, given $[r_1, r_2, r_3, r_4, \ldots, r_T]$ we want to privately release

$$\left[r_1, \sum_{i=1}^{2} r_i, \sum_{i=1}^{3} r_i, \sum_{i=1}^{4} r_i, \ldots, \sum_{i=1}^{T} r_i \right].$$

An interesting sophisticated differentially private algorithm for this problem is the *two-level counter* from Chan, et al. [14]; we call this algorithm smartsum.

At a high level, this algorithm groups the input list into blocks of length q, and adds Laplace noise to the sum for each block. More concretely, to compute a running sum from 1 to t with t a multiple of q, we simply add together the first t/q block sums. If t is not a multiple of q, say $t = qs + r$ with $r < q$, we take the first s block sums and add a noised version of each of the r remaining elements.

For an example, suppose we take $q = 3$ and T is a multiple of 3. For brevity, let us use the notation $\mathsf{L}(r)$ to describe the result of the application of Laplace, for a fixed value ϵ to r. Then, the output of smartsum is

$$\left[\mathsf{L}(r_1), \mathsf{L}(r_1) + \mathsf{L}(r_2), \mathsf{L}\left(\sum_{i=1}^{3} r_i\right), \right.$$
$$\left. \mathsf{L}\left(\sum_{i=1}^{3} r_i\right) + \mathsf{L}(r_4), \ldots, \sum_{j=0}^{T/3} \mathsf{L}\left(\sum_{i=1}^{3} r_{3j+i}\right) \right].$$

To informally argue privacy, observe that if we run the Laplace mechanism on each individual entry, there is no privacy cost for the indices where the adjacent databases are the same. So, the privacy analysis for smartsum is straightforward: changing an input element will change exactly two noisy sums—the sum for the block containing i, and the noisy version of i—and each noisy sum that can change requires ϵ privacy budget, since we are using the Laplace mechanism with parameter ϵ. Thus, smartsum is 2ϵ-private.

The full program, together with the transformation into a synchronized product program, is presented in Fig. 6. The formal verification of the 2ϵ-differential privacy follows the argument above. The pre-condition states that the two input databases are adjacent, while the post-condition requires equality on the outputs and bounds the accumulated privacy budget by 2ϵ.

The interesting part for our verification is the while loop. Indeed, this requires a loop invariant to keep track of the privacy budget, which depends on whether the differing entry has been processed or not. As mentioned in the previous section, this invariant does not fit the apRHL while rule of Fig. 5: to deal with this example, Barthe et al. [11] introduce a generalized while rule able to perform refined analysis depending on a predicate preserved across iterations. In contrast, here we do not require any special verification rule: the standard while rule from Hoare logic suffices.

More precisely, we apply the Hoare while rule with the invariant:

$$\begin{array}{l} \mathsf{adjacent}(l_1, l_2) \wedge out_1 = out_2 \wedge next_1 = next_2 \wedge n_1 = n_2 \wedge \\ |c_1 - c_2| \leq 1 \wedge (l_1 \neq l_2 \Rightarrow \mathrm{v}_\epsilon = 0) \wedge \\ (c_1 \neq c_2 \Rightarrow l_1 = l_2 \wedge \mathrm{v}_\epsilon \leq \epsilon) \wedge (l_1 = l_2 \rightarrow \mathrm{v}_\epsilon \leq 2\,\epsilon) \end{array}$$

Notice from the invariant that if the accumulators c_1 and c_2 differ we have $l_1 = l_2$. This corresponds to the fact that the differing entry has been processed and so the remaining database entries coincide. Also, if this is the case then the privacy budget of $2\,\epsilon$ has been already consumed.

The verification of this invariant proceeds by case analysis. We have three cases: a) the differing entry has not been processed yet and will not be processed in the following iteration, b) the differing entry has not been processed yet but is going to be processed in the next iteration, and c) the differing entry has already been processed, in which case there is no more privacy budget consumption.

B. Multiplicative Weights Exponential Mechanism

While answering queries on a database with the Laplace mechanism is a simple way to guarantee privacy, the added noise quickly renders the results useless as the number of queries grows. To handle larger collections of queries, there has been much research on sophisticated algorithms based on learning theory.

One such scheme is *Iterative Database Construction (IDC)*, due to Gupta et al. [26]. The basic idea is simple: given a database $\hat{d}$, the algorithm gradually builds a *synthetic database* that approximates the original database. The synthetic database is built over several rounds; after some fixed number of rounds, the synthetic database is released and used to answer all queries.

The essence of the algorithm is the computation that it performs at each round. Let $\mathcal{Q}$ be a collection of queries that we want to answer and let d^i be the synthetic database computed at round i. During round $i+1$, the algorithm selects a query $q \in \mathcal{Q}$ with high error; that is, a query where the current approximate database d^i and the true database $\hat{d}$ give very different answers. This selection is done in a differentially private way. Next, the algorithm computes a noisy version v of q evaluated on the true database $\hat{d}$. Again, this step must be differentially private. Finally, q, v and the current database d^i approximation are fed into an update algorithm, which generates the next approximation d^{i+1} of the synthetic database (hopefully performing better on q).

```
$next \leftarrow 0; n \leftarrow 0; c \leftarrow 0;$
while $0 <$ length $l$ do
  if length $l$ mod $q = 0$ then
    $x \leftarrow$ Lap$_\epsilon(c +$ hd $l)$;
    $n \leftarrow x + n$;
    $next \leftarrow n$;
    $c \leftarrow 0$;
    $out \leftarrow next :: out$;
  else
    $x \leftarrow$ Lap$_\epsilon($hd $l)$;
    $next \leftarrow next + x$;
    $c \leftarrow c +$ hd $l$;
    $out \leftarrow next :: out$;
  $l \leftarrow$ tl $l$;
return $out$;
```

(a) Original probabilistic algorithm

```
v$_\epsilon \leftarrow 0$; $next_1 \leftarrow 0$; $next_2 \leftarrow 0$;
$n_1 \leftarrow 0$; $n_2 \leftarrow 0$; $c_1 \leftarrow 0$; $c_2 \leftarrow 0$;
assert $((0 <$ length $l_1) \Leftrightarrow (0 <$ length $l_2))$;
while $0 <$ length $l_1$ do
  assert $(($length $l_1$ mod $q = 0) \Leftrightarrow ($length $l_2$ mod $q = 0))$;
  if length $l_1$ mod $q = 0$ then
    $(x_1, x_2) \leftarrow$ Lap$^\diamond_\epsilon(c_1 +$ hd $l_1, c_2 +$ hd $l_2))$;
    $n_1 \leftarrow x_1 + n_1$; $n_2 \leftarrow x_2 + n_2$;
    $next_1 \leftarrow n_1$; $next_2 \leftarrow n_2$;
    $c_1 \leftarrow 0$; $c_2 \leftarrow 0$;
    $out_1 \leftarrow next_1 :: out_1$; $out_2 \leftarrow next_2 :: out_2$;
  else
    $(x_1, x_2) \leftarrow$ Lap$^\diamond_\epsilon($hd $l_1,$ hd $l_2))$;
    $next_1 \leftarrow next_1 + x_1$; $next_2 \leftarrow next_2 + x_2$;
    $c_1 \leftarrow c_1 +$ hd $l_1$; $c_2 \leftarrow c_2 +$ hd $l_2$;
    $out_1 \leftarrow next_1 :: out_1$; $out_2 \leftarrow next_2 :: out_2$;
  $l_1 \leftarrow$ tl $l_1$; $l_2 \leftarrow$ tl $l_2$;
return $(out_1, out_2)$;
```

(b) Synchronized non-probabilistic product

Fig. 6: smartsum algorithm

The key point is that in many cases, this iterative procedure will provably find an approximation with low error on *all* queries in $\mathcal{Q}$ in a small number of steps. Hence, we can run IDC for a small number of steps, and release the final database approximation as the output. Queries in $\mathcal{Q}$ can then be evaluated on this output for an accurate estimate of the true answer to the query.

IDC is actually a family of algorithms parameterized by an algorithm to privately find a high-error query (called the *private distinguisher*), and the update function (called the *database update algorithm*). For concreteness, let us consider one well-studied instantiation, the *Multiplicative Weights Exponential Mechanism* (MWEM) algorithm originally due to Hardt and Rothblum [28] and experimentally evaluated by Hardt et al. [27].

MWEM uses the exponential mechanism to privately select a query with high error—the quality score of a query q to be maximized is the error of the query, i.e., the absolute difference between q evaluated on the approximate database d^i and q evaluated on the true database $\hat{d}$. The update function applies the multiplicative weights update [3] to adjust the approximation to perform better on the mishandled query. This step is non-private: it does not touch the private data directly. Hence, we do not concern ourselves with the details here, and treat the update step as a black box. (The reader can find further details in Hardt et al. [27].) The full program, together with the transformation into a synchronized product program, is presented in Fig. 7.

We briefly comment on the program. We let d^i denote the i-th iteration of the synthetic database, and $\hat{d}$ denote the true database. Initially the synthetic database d^0 is set to some default value def. Then we define the score function s^i that takes as inputs a database D and a query Q and returns the error of the query Q on the current approximation d^i compared to D. We then apply the exponential mechanism to the true database $\hat{d}$ with the score function s^i, and we call the result q^i. We then evaluate q^i on the real database, and add Laplace noise; we call the result a^i. Finally, we apply the update function to obtain the next iteration d^{i+1} of the synthetic database. Once the number of rounds is exhausted, we return the last computed synthetic databases.

For the privacy proof, we assume that all queries in $\mathcal{Q}$ are 1-sensitive. Note that we run T iterations of MWEM; by the composition theorem, it is sufficient to analyze the privacy budget consumed by each iteration. Each iteration, we select a query with the exponential mechanism with privacy parameter ϵ, and we estimate the true answer of this query with the Laplace mechanism, parameter ϵ. By the composition theorem (Theorem 3), the whole algorithm is private with parameter $2 \cdot T \cdot \epsilon = 2T\epsilon$, as desired. The proof can be transcripted directly into Hoare logic using self-products; we take as pre-condition adjacency of the two databases, and use adjacency to conclude that the sensitivity of the score function s_i is 1 at each iteration.

C. *Propose-Test-Release*

The examples we have considered so far all rely on the composition theorem. While this is a quite powerful and useful theorem, not all algorithms use composition. In this section, we consider one such example: the *Propose-Test-Release* (PTR) framework [20], [38]. PTR is also an example of an (ϵ, δ)-differentially private mechanism for $\delta > 0$.

The motivation comes from private release of statistics that are sometimes, but not always, very sensitive. For example, suppose our database is an ordered list of numbers between 0 and 1000, and suppose we want to release the median element of the database. This can be highly sensitive: consider the database $[0, 0, 1000]$ with median 0. Adding a record 1000 to the database would lead to a large change in the median (now 500, if we average the two elements closest to the median when the database has even size). However, many

```
$i \leftarrow 0$;
$d^0 \leftarrow \mathrm{def}$;
while $i < T$ do
    $s^i \leftarrow \lambda D\ Q.\ |Q(d^i) - Q(D)|$
    $q^i \leftarrow \mathsf{Exp}_\epsilon\,(s^i, \hat{d})$;
    $a^i \leftarrow \mathsf{Lap}_\epsilon\,(q^i\,\hat{d})$;
    $d^{i+1} \leftarrow \mathsf{update}\,(d^i, a^i, q^i)$;
    $i \leftarrow i + 1$;
return $d^T$;
```

(a) Original probabilistic algorithm

```
$\mathsf{v}_\epsilon \leftarrow 0;\ i_1 \leftarrow 0;\ i_2 \leftarrow 0$;
$d_1^0 \leftarrow \mathrm{def}; d_2^0 \leftarrow \mathrm{def}$;
assert $(i_1 < T \Leftrightarrow i_2 < T)$;
while $i_1 < T$ do
    $s_1^i \leftarrow \lambda D\ Q.\ |Q(d_1^i) - Q(D)|$;
    $s_2^i \leftarrow \lambda D\ Q.\ |Q(d_2^i) - Q(D)|$;
    $(q_1^i, q_2^i) \leftarrow \mathsf{Exp}_\epsilon^\diamond(s_1^i, \hat{d}_1, s_2^i, \hat{d}_2)$;
    $(a_1^i, a_2^i) \leftarrow \mathsf{Lap}_\epsilon^\diamond(q_1^i(\hat{d}_1), q_2^i(\hat{d}_2))$;
    $d_1^{i+1} \leftarrow \mathsf{update}\,(d_1^i, a_1^i, q_1^i)$;
    $d_2^{i+1} \leftarrow \mathsf{update}\,(d_2^i, a_2^i, q_2^i)$;
    $i_1 \leftarrow i_1 + 1$;
    $i_2 \leftarrow i_2 + 1$;
    assert $(i_1 < T \Leftrightarrow i_2 < T)$;
return $(d_1^T, d_2^T)$;
```

(b) Synchronized non-probabilistic product

Fig. 7: MWEM algorithm

other databases have low sensitivities: for $[0, 10, 10, 1000]$, the median will remain unchanged (at 10) no matter what element we add or remove from the database. We may hope that we can privately compute the median in this second case with much less noise than needed for the first case. More generally, the second database is quite *stable*—all adjacent databases have the same median value. In contrast, the first database is *instable*—adjacent databases may have wildly different median values. With this example in mind, we now explain the general PTR framework.

Suppose we want to privately release the result of a query q evaluated on a database d. We assume that databases are taken from a set $\mathcal{D}$ and that there exists a notion of distance Δ on $\mathcal{D}$. First, we estimate the *distance to instability*—that is, the largest distance x such that $q(d) = q(d')$ for all databases d' at distance x or less from d. Since this a 1-sensitive function (moving to a neighboring database can change the distance to instability by at most 1), we can release this distance privately using the Laplace mechanism (say, with parameter ϵ). Call the result y. Now, we compare y to a threshold t (to be specified later). If y is less than the threshold, we output $q(d)$ with no noise. If y is greater than the threshold, we output a default value $\perp$. The program is given in Fig. 8.

```
$x \leftarrow \mathsf{DistToInstability}\,(q, d)$;
$y \leftarrow \mathsf{Lap}_{\,\epsilon}\ x$;
if $(|y| > \log(2/\delta)/(2\epsilon))$
    return $(q\,d)$;
else
    return $(\perp)$;
```

Fig. 8: PTR algorithm

The privacy of the algorithm can be informally justified in two parts. First, suppose that instead of outputting $q(d)$ or $\perp$, we simply output which branch the program took. This is ϵ-differentially private: computing y is ϵ-differentially private (via the Laplace mechanism), and the resulting branch is a post-processing of y. Hence, we can assume that the same branch is taken in both executions.

Second, we can conclude that the original program (outputting $q(d)$ or $\perp$) is (ϵ, δ)-differentially private if for any adjacent databases d and d' with $q(d) \neq q(d')$, the first branch is taken with probability at most δ. By properties of the Laplace mechanism, we can set the threshold t large enough so that with probability at least $1 - \delta$, the first branch is only taken if x is strictly positive. In this case we can conclude $q(d) = q(d')$, since $q(d) \neq q(d')$ implies that x is 0 on both executions. So, we can safely release $q(d) = q(d')$ with no noise. Of course, if the second branch is taken, then it is also safe to release $\perp$ in both runs.

More formally, the proof of (ϵ, δ)-differential privacy for PTR rests on two properties of the Laplace mechanism: the privacy property captured by Theorem 1 and the accuracy property captured by Lemma 1.

Fig. 9 presents the proof of PTR using the synchronized product program—the code is interleaved with some of the pre- and post-conditions. The proof uses the accuracy property of the Laplace mechanism and the properties of the distance to instability that we give as specifications in Fig. 10. For simplicity, we treat distance to instability as an abstract procedure; however, it can be implemented as a loop over all databases, in which case the specification can be proved. The soundness of the accuracy specification for the Laplace mechanism is shown in the appendix.

D. *Vertex cover*

A vertex cover for a graph $g = (N, E)$ is a set S of nodes such that for every edge $(t, u) \in E$, either $t \in S$ or $u \in S$. The minimum vertex cover is the problem of finding a vertex cover of a minimum size. Gupta et al. [25] study the problem of privately computing a minimum vertex cover in a setting where the nodes of the graph are public, but its edges are private. Since a vertex cover leaks information about vertices (for instance, any two nodes that are not in the vertex cover are certainly not connected by an edge), their algorithm outputs an enumeration of the nodes of the graph, from which a vertex cover can be recomputed efficiently from the knowledge of the set E. Their algorithm is challenging to verify because rather

$$\vdash (y_1, y_2) \leftarrow \mathsf{Lap}^\diamond_\epsilon(x_1, x_2) : x_1 = x_2 \wedge \mathrm{v}_\delta = \hat{\delta} \Longrightarrow y_1 = y_2 \wedge |y_1 - x_1| \leq \log(2/\delta)/(2\epsilon) \wedge \mathrm{v}_\delta = \hat{\delta} + \delta$$

$$\vdash x_1 \leftarrow \mathsf{DistToInstability}\,(q, d_1); x_2 \leftarrow \mathsf{DistToInstability}\,(q, d_2) : \Delta(d_1, d_2) \leq 1 \Longrightarrow q(d_1) = q(d_2) \vee x_1 = x_2 = 0$$

Fig. 10: Accuracy specification for the Laplace mechanism, and specification for distance to instability.

```
$\{\Delta(d_1, d_2) \leq 1\}$
$\mathrm{v}_\epsilon \leftarrow 0$;
$\mathrm{v}_\delta \leftarrow 0$;
$x_1 \leftarrow$ DistToInstability $(q, d_1)$;
$x_2 \leftarrow$ DistToInstability $(q, d_2)$;
$\left\{ \begin{array}{l} (q(d_1) = q(d_2) \vee x_1 = x_2 = 0) \\ \wedge \mathrm{v}_\epsilon = 0 \wedge \mathrm{v}_\delta = 0 \end{array} \right\}$
$(y_1, y_2) \leftarrow \mathsf{Lap}^\diamond_\epsilon(x_1, x_2)$;
$\left\{ \begin{array}{l} (q(d_1) = q(d_2)) \vee (x_1 = x_2 = 0 \\ \wedge |y_1 - x_1| \leq \log(2/\delta)/(2\epsilon)) \\ \wedge y_1 = y_2 \wedge \mathrm{v}_\epsilon \leq \epsilon \wedge \mathrm{v}_\delta \leq \delta \end{array} \right\}$
    assert $(|y_1| > \log(2/\delta)/(2\epsilon) \Leftrightarrow |y_2| > \log(2/\delta)/(2\epsilon))$;
if $(|y_1| > \log(2/\delta)/(2\epsilon))$
    $\{\ q(d_1) = q(d_2) \wedge \mathrm{v}_\epsilon \leq \epsilon \wedge \mathrm{v}_\delta \leq \delta\ \}$
    return $(q(d_1), q(d_2))$;
    $\{\ \mathsf{out}_1 = \mathsf{out}_2 \wedge \mathrm{v}_\epsilon \leq \epsilon \wedge \mathrm{v}_\delta \leq \delta\ \}$
else
    $\{\ \mathrm{v}_\epsilon \leq \epsilon \wedge \mathrm{v}_\delta \leq \delta\ \}$
    return $(\perp, \perp)$;
    $\{\ \mathsf{out}_1 = \mathsf{out}_2 \wedge \mathrm{v}_\epsilon \leq \epsilon \wedge \mathrm{v}_\delta \leq \delta\ \}$
```

Fig. 9: Proof of Propose-Test-Release

```
$n \leftarrow |E|$;
$out \leftarrow []$;
while $g \neq \emptyset$ do
    $v \leftarrow \mathsf{choose}_{\epsilon,n}(g)$;
    $out \leftarrow v :: out$;
    $g \leftarrow g \setminus \{v\}$;
return $out$;
```

Fig. 11: Minimum vertex cover

than relying on mechanisms, it achieves privacy by sampling according to a suitable noisy distribution choose. The code of the algorithm is shown in Fig. 11.

We say that two graphs g_1 and g_2 are adjacent if they differ at most in one edge $\langle t, u\rangle$. By defining choose as

$$\Pr\left[v \leftarrow \mathsf{choose}_{\epsilon,n}(g) : v = v'\right] \propto \left(d_{E,V}(v') + \frac{4}{\epsilon}\sqrt{\frac{n}{|E|}}\right)$$

where $g = (E, V)$ and n is a given parameter, one obtains an $(\epsilon, 0)$-differentially private algorithm with respect to the adjacency relation as defined above.

One can prove differential privacy using an extension of apRHL with ad hoc rules (see Appendix D for details).

We now consider the formal verification of the vertex cover algorithm using self-products. We first extend the definition of self-product to choose. Then, there are two cases to consider: $g_2 = g_1 \cup \{\langle u, t\rangle\}$ and $g_1 = g_2 \cup \{\langle u, t\rangle\}$. In the first case, we can use the first Hoare specification from Fig. 12. In the second case, we use the second and third specifications from Fig. 12. Using these specifications, it is possible to verify that the self-product of the vertex cover algorithm satisfies the Hoare specification of Theorem 5. However, we have not yet been able to extend the proof of Theorem 5 to deal with the choose self-product.

E. Formal verification of the examples

The examples above (with the exception of vertex cover) have been formally verified. For each example, we have built the corresponding self-product program, and verified this result using the non-probabilistic and non-relational Hoare logic rules available in the EasyCrypt [6] framework. As described above, we have used non-probabilistic axiomatic specifications for the primitives. Apart from the axiomatic specification, and the code for the program and the self-product construction, the longest Hoare logic verification proof (for MWEM) consists of about 50 lines of code. This demonstrates the simplicity offered by the self-product construction. The code for these examples (and others) is available online [1].

VI. Related work

Differential privacy, first proposed by Blum et al. [13] and formally defined by Dwork et al. [21], has been an area of intensive research in the last decade. We have touched on a handful of private algorithms, including algorithms for computing running sums [14], [22] (part of a broader literature on streaming privacy), answering large classes of queries [28], [27] (part of a broader literature on learning-theoretic approaches to data privacy), the Propose-Test-Release framework for answering stable queries in a noiseless way [20], [38], and private combinatorial optimization [25]. We refer readers interested in a more comprehensive treatment to the excellent surveys by Dwork [18], [19].

Verifying differential privacy: Several tools have been proposed for providing formal verification of the differential privacy guarantee; we can roughly classify them by the verification approach they use. PINQ [31] provides an encapsulation for LINQ—an SQL-like language embedded in C#—tracking at runtime the privacy budget consumption, and aborting the computation when the budget is exhausted. Airavat [35] combines a similar runtime monitor with access control in a MapReduce framework. While PINQ is restricted to ϵ-differential privacy, Airavat can handle also approximate differential privacy using a runtime monitor for δ.

$$\vdash (v_1, v_2) \leftarrow \mathsf{choose}^{\diamond}_{\epsilon,n}(g_1, g_2) : g_1 \cup \{\langle u, t\rangle\} = g_2 \wedge \mathrm{v}_\epsilon = \epsilon_0 \Longrightarrow v_1 = v_2 \wedge \mathrm{v}_\epsilon = \epsilon_0 + \epsilon / \left(2\sqrt{n}\sqrt{|g_1|}\right)$$

$$\vdash (v_1, v_2) \leftarrow \mathsf{choose}^{\diamond}_{\epsilon,n}(g_1, g_2) : g_1 = g_2 \cup \{\langle u, t\rangle\} \wedge \mathrm{v}_\epsilon = \epsilon_0 \Longrightarrow (v \neq t \wedge v \neq u) \wedge \mathrm{v}_\epsilon = \epsilon_0$$

$$\vdash (v_1, v_2) \leftarrow \mathsf{choose}^{\diamond}_{\epsilon,n}(g_1, g_2) : g_1 = g_2 \cup \{\langle u, t\rangle\} \wedge \mathrm{v}_\epsilon = \epsilon_0 \Longrightarrow (v = t \vee v = u) \wedge \mathrm{v}_\epsilon = \epsilon_0 + \epsilon/4$$

Fig. 12: Hoare specifications for $\mathsf{choose}^{\diamond}$

Another approach is based on linear type systems. Fuzz [34] and DFuzz [24] use a type-based approach for inferring and checking the sensitivity of functional programs. This sensitivity analysis combined with the use of trusted probabilistic primitives provides the differential privacy guarantee. Interestingly, this type-based approach can be combined with type systems for cryptographic protocols to verify differential privacy for distributed protocols [23]. All these systems provide automatic verification of differential privacy. However, they fail to verify all the examples that we can handle, like advanced sum statistics [14] and the Propose-Test-Release framework [20]. Moreover, so far they can address only pure differential privacy, where $\delta = 0$.

Tschantz, et al. [39] consider a verification framework for interactive private programs, where the algorithm can receive new input and produce multiple outputs over a series of steps. They follow an approach similar to ours by verifying the correct use of differentially private primitives. However, their programs are well-modeled by probabilistic I/O-automata, and they provide a proof technique based on probabilistic bisimulation. Also, their method is currently limited to pure differential privacy.

Finally, CertiPriv [11] and EasyCrypt [6] use custom relational logics to verify differential privacy. These systems are very expressive: they supports general (ϵ, δ)-differential privacy, they can verify privacy for mechanisms like the Laplace and the Exponential mechanism, and they can capture advanced examples that go beyond mechanisms and composition, like the private vertex cover algorithm of Gupta et al. [25]. The difficulty with their approach is that it relies on a customized and complex logic. Moreover, ad hoc rules for loops are required for many advanced examples.

Verifying 2-safety properties: Beyond differential privacy, there is a large body of literature on verifying 2-safety properties. Our work is most closely related to deductive methods based on program logics; more precisely, approaches that reduce 2-safety of a program c to safety of a program c' built from c. Such approaches include *self-composition* [7], *product programs* [40], and *type-directed product programs* [37]. These approaches are subsumed by work by Barthe et al. [4], [5].

Another alternative is to reason directly on two programs (or two executions of the same program) using relational program logics such as Benton's relational Hoare logic [12], or specialized relational logics, e.g., for information flow [2]. CertiCrypt [9], and EasyCrypt [8], [6], are computer-aided tools that support relational reasoning about probabilistic programs and have been used to prove security of cryptographic constructions and computational differential privacy of protocols. For such applications, reasoning about structurally different programs is essential.

Chaudhuri et al. [15] develop an automated method for analyzing the continuity and the robustness of programs. Robustness is a 2-safety property that is very similar to sensitivity as used in differential privacy. An interesting aspect of their work is that their analysis is able to reason about two *unsynchronized* pairs of executions; that is, pairs of executions that may have different control flow.

Verification of hyperproperties: Developing general verification methods for hyperproperties remains a challenge; however, there have been some recent proposals in this direction (e.g., [29], [30], [33]).

Other work: There is an extensive body of work on deductive verification of non-probabilistic and probabilistic programs, as well as many works that consider product constructions of Labeled Transition Systems; summarizing this large literature is beyond the scope of this paper.

VII. Conclusion

We have proposed a program transformation that reduces proving (ϵ, δ)-differential privacy of a probabilistic program to proving a safety property of a deterministic transformed program. The method applies to all standard examples where privacy is achieved through mechanisms and composition theorems; on the other hand, differentially private algorithms based on ad hoc output perturbation, such as the differentially private vertex cover algorithm [25], are more difficult to handle. In particular, they fall outside the scope of Theorem 5 which proves the soundness of our approach. Our method is particularly suited for reasoning about differential privacy, because the transformed program can be analyzed with standard verification tools. Our method can also be extended to reason about probabilistic non-interference, at the cost of targeting an assertion language that supports existential quantification over functions. Directions for further work include extending the scope of Theorem 5 to deal with more complex examples, like vertex cover. On a more practical side, it would be interesting to implement our transformation for a realistic setting, for instance modeling the PINQ language [31].

References

[1] Proving differential privacy in hoare logic; supplementary code for the examples verified in EasyCrypt, 2014. http://www.easycrypt.info/selfproduct/selfproduct.tar.gz.

[2] Torben Amtoft and Anindya Banerjee. Information flow analysis in logical form. In *11th International Symposium on Static Analysis, SAS 2004*, volume 3148 of *Lecture Notes in Computer Science*, pages 100–115, Heidelberg, 2004. Springer.

[3] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012.

[4] Gilles Barthe, Juan Manuel Crespo, and César Kunz. Relational verification using product programs. In Michael Butler and Wolfram Schulte, editors, *FM*, volume 6664 of *Lecture Notes in Computer Science*, pages 200–214. Springer, 2011.

[5] Gilles Barthe, Juan Manuel Crespo, and César Kunz. Beyond 2-safety: Asymmetric product programs for relational program verification. In Sergei N. Artëmov and Anil Nerode, editors, *LFCS*, volume 7734 of *Lecture Notes in Computer Science*, pages 29–43. Springer, 2013.

[6] Gilles Barthe, George Danezis, Benjamin Grégoire, César Kunz, and Santiago Zanella Béguelin. Verified computational differential privacy with applications to smart metering. In *CSF*, pages 287–301. IEEE, 2013.

[7] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In Jonathan Herzog, editor, *CSFW*, pages 100–114. IEEE Computer Society, 2004.

[8] Gilles Barthe, Benjamin Grégoire, Sylvain Heraud, and Santiago Zanella Béguelin. Computer-aided security proofs for the working cryptographer. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 71–90. Springer, 2011.

[9] Gilles Barthe, Benjamin Grégoire, and Santiago 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.

[10] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In John Field and Michael Hicks, editors, *POPL*, pages 97–110. ACM, 2012.

[11] Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. *ACM Trans. Program. Lang. Syst.*, 35(3):9, 2013.

[12] Nick 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.

[13] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the SuLQ framework. In *ACM SIGACT–SIGMOD–SIGART Symposium on Principles of Database Systems (PODS), Baltimore, Maryland*, pages 128–138, 2005.

[14] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Private and continual release of statistics. *ACM Transactions on Information and System Security*, 14(3):26, 2011.

[15] Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. Continuity analysis of programs. In Manuel V. Hermenegildo and Jens Palsberg, editors, *POPL*, pages 57–70. ACM, 2010.

[16] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *21st IEEE Computer Security Foundations Symposium, CSF 2008*, pages 51–65, Los Alamitos, 2008. IEEE Computer Society.

[17] Ádám Darvas, Reiner Hähnle, and David Sands. A theorem proving approach to analysis of secure information flow. In Dieter Hutter and Markus Ullmann, editors, *SPC*, volume 3450 of *Lecture Notes in Computer Science*, pages 193–209. Springer, 2005.

[18] Cynthia Dwork. Differential privacy. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP (2)*, volume 4052 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2006.

[19] Cynthia Dwork. Differential privacy: A survey of results. In Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation*, volume 4978 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin Heidelberg, 2008.

[20] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In Michael Mitzenmacher, editor, *STOC*, pages 371–380. ACM, 2009.

[21] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *IACR Theory of Cryptography Conference (TCC), New York, New York*, pages 265–284, 2006.

[22] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. Differential privacy under continual observation. In *ACM SIGACT Symposium on Theory of Computing (STOC), Cambridge, Massachusetts*, pages 715–724, 2010.

[23] Fabienne Eigner and Matteo Maffei. Differential privacy by typing in security protocols. In *CSF*, pages 272–286. IEEE, 2013.

[24] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. Linear dependent types for differential privacy. In Roberto Giacobazzi and Radhia Cousot, editors, *POPL*, pages 357–370. ACM, 2013.

[25] Anupam Gupta, Katrina Ligett, Frank McSherry, Aaron Roth, and Kunal Talwar. Differentially private combinatorial optimization. In *ACM–SIAM Symposium on Discrete Algorithms (SODA), Austin, Texas*, pages 1106–1125, 2010.

[26] Anupam Gupta, Aaron Roth, and Jonathan Ullman. Iterative constructions and private data release. In *IACR Theory of Cryptography Conference (TCC), Taormina, Italy*, pages 339–356, 2012.

[27] Moritz Hardt, Katrina Ligett, and Frank McSherry. A simple and practical algorithm for differentially private data release. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *NIPS*, pages 2348–2356, 2012.

[28] Moritz Hardt and Guy N. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *FOCS*, pages 61–70. IEEE Computer Society, 2010.

[29] Andrew K. Hirsch and Michael R. Clarkson. Belief semantics of authorization logic. In Sadeghi et al. [36], pages 561–572.

[30] Máté Kovács, Helmut Seidl, and Bernd Finkbeiner. Relational abstract interpretation for the verification of 2-hypersafety properties. In Sadeghi et al. [36], pages 211–222.

[31] Frank McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proc. SIGMOD*, 2009.

[32] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103. IEEE Computer Society, 2007.

[33] Dimiter Milushev and Dave Clarke. Incremental hyperproperty model checking via games. In Hanne Riis Nielson and Dieter Gollmann, editors, *NordSec*, volume 8208 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013.

[34] Jason Reed and Benjamin C. Pierce. Distance makes the types grow stronger: a calculus for differential privacy. In Paul Hudak and Stephanie Weirich, editors, *ICFP*, pages 157–168. ACM, 2010.

[35] Indrajit Roy, Srinath Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. Airavat: Security and privacy for MapReduce. In *USENIX Symposium on Networked Systems Design and Implementation (NDSI), San Jose, California*, 2010.

[36] Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors. *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*. ACM, 2013.

[37] Tachio Terauchi and Alex 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.

[38] Abhradeep Thakurta and Adam Smith. Differentially private feature selection via stability arguments, and the robustness of the lasso. In Shai Shalev-Shwartz and Ingo Steinwart, editors, *COLT*, volume 30 of *JMLR Proceedings*, pages 819–850. JMLR.org, 2013.

[39] Michael Carl Tschantz, Dilsun Kaynar, and Anupam Datta. Formal verification of differential privacy for interactive systems (extended abstract). *Electronic Notes in Theoretical Computer Science*, 276(0):61 – 79, 2011. Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics (MFPS XXVII).

[40] Anna Zaks and Amir Pnueli. Covac: Compiler validation by program analysis of the cross-product. In *15th International Symposium on Formal Methods, FM 2008*, volume 5014 of *Lecture Notes in Computer Science*, pages 35–51, Heidelberg, 2008. Springer.

Appendix A
Definitions

A. Semantics of the target language

The semantics of the non-deterministic target language is defined as a function from a memory to a set of memories. In order to distinguish the failure of assert statements from non-terminating while loops, we lift the domain $\mathcal{P}(\mathcal{M})$ with a $\perp$ element:

$$
\begin{array}{lcl}
[\![\mathsf{skip}]\!]\, m & = & \{m\} \\
[\![c_1;\, c_2]\!]\, m & = & \bigcup_{m' \in [\![c_1]\!]\, m} [\![c_2]\!]\, m \\
[\![x \leftarrow e]\!]\, m & = & \mathbb{1}_{m\{[\![e]\!]_{\mathcal{E}}\, m/x\}} \\
[\![\mathsf{assert}\,(\varphi)]\!]\, m & = & \textbf{if } ([\![\varphi]\!]_{\mathcal{E}}\, m) \textbf{ then } \{m\} \textbf{ else } \perp \\
[\![(x_1, x_2) \leftarrow \mathsf{Lap}^{\diamond}_{\epsilon}(e_1, e_2)]\!]\, m & = & \bigcup_v m\,\{v/x_1\}\,\{v/x_2\}\,\{\mathtt{v}_\epsilon + |e_1 - e_2|\epsilon/\mathtt{v}_\epsilon\} \\
[\![(x_1, x_2) \leftarrow \mathsf{Exp}^{\diamond}_{\epsilon}(s_1, e_1, s_2, e_2)]\!]\, m & = & \textbf{if } ([\![s_1 = s_2]\!]_{\mathcal{E}}\, m = \mathsf{true}) \textbf{ then} \\
 & & \quad \bigcup_v m\,\{v/x_1\}\,\{v/x_2\}\,\{\mathtt{v}_\epsilon + |e_1 - e_2|\epsilon/\mathtt{v}_\epsilon\} \\
 & & \textbf{else } \perp \\
[\![\mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2]\!]\, m & = & \textbf{if } ([\![e]\!]_{\mathcal{E}}\, m = \mathsf{true}) \textbf{ then } ([\![c_1]\!]\, m) \textbf{ else } ([\![c_2]\!]\, m) \\
[\![\mathsf{while}\ e\ \mathsf{do}\ c]\!]\, m & = & \bigsqcup \hat{w}_i\, m
\end{array}
$$

$$
\text{where} \quad
\begin{array}{lcl}
\hat{w}_0\, m & = & \emptyset \\
\hat{w}_{i+1}\, m & = & \textbf{if } ([\![e]\!]_{\mathcal{E}}\, m = \mathsf{true}) \textbf{ then } \bigcup_{m' \in ([\![c]\!]\, m)} \hat{w}_i\, m' \textbf{ else } \{m\}
\end{array}
$$

where $\bigcup_{m \in \perp} f\, m$ is defined as $\perp$ for any f.

Appendix B
Auxiliary lemmas

The following is an auxiliary result used in the proof of correctness of the method based on self-products.

Lemma 3: Suppose that for all memories m_1, m_2 such that $m_1\, \Psi\, m_2$ we have that c is terminating in m_1 and m_2, and $([\![c]\!]\, m_1)\, \Phi_{\langle \epsilon, \delta \rangle}\, ([\![c]\!]\, m_2)$. Then, for every memory distributions μ_1, μ_2 such that $\mu_1\, \Psi_{\langle \epsilon', \delta' \rangle}\, \mu_2$ we have

$$([\![c]\!]^\star\, \mu_1)\, \Phi_{\langle \epsilon + \epsilon', \delta + \delta' \rangle}\, ([\![c]\!]^\star\, \mu_2)$$

The following is another auxiliary result used in the proof of correctness.

Lemma 4: For all memories m_1, m_2 such that $m_1\, \Psi\, m_2$ we have that $\mathbb{1}_{m_1}\, \Psi_{\langle 0,0 \rangle}\, \mathbb{1}_{m_2}$.

Proof: We can take as witness $\hat{\mu} = \mathbb{1}_{m_1, m_2}$. ■

Lemma 5: Suppose that for m_1, m_2 such that $m_1\, \Psi\, m_2$ we have that $(\mu_1\, m_1)\, \Phi_{\langle \epsilon, \delta \rangle}\, (\mu_2\, m_2)$. Then,

$$((\lambda v. \mathbb{1}_{m_1\{v/x\}})^\star\, \mu_1)\, \Phi_{\langle \epsilon, \delta \rangle}\, ((\lambda v. \mathbb{1}_{m_2\{v/x\}})^\star\, \mu_2)$$

Proof: By Lemma 4 and Lemma 3. ■

The proof of the next two auxiliary lemmas are presented in the work in apRHL [11].

Lemma 6: Given a relation S that is *preserved* by c, i.e. such that:

$$\forall m_1, m_2.\ (m_1, m_2) \in S \Rightarrow (\forall m_1', m_2'. ([\![c]\!]\, m_1\, m_1' \neq 0 \wedge [\![c]\!]\, m_2\, m_2' \neq 0 \Rightarrow (m_1', m_2') \in S))$$

If

$$\forall m_1, m_2.\ (m_1, m_2) \in R \Rightarrow ([\![c]\!]\, m_1)\, Q_{\langle \epsilon, \delta \rangle}\, ([\![c]\!]\, m_2)$$

then

$$\forall m_1, m_2.\ (m_1, m_2) \in (R \cap S) \Rightarrow ([\![c]\!]\, m_1)\, (Q \cap S)_{\langle \epsilon, \delta \rangle}\, ([\![c]\!]\, m_2)$$

Lemma 7: For all distribution expressions μ_1, μ_2, if

$$\Delta_\epsilon([\![\mu_1]\!]\, m_1, [\![\mu_2]\!]\, m_2) \leq \delta$$

then

$$([\![x \overset{\$}{\leftarrow} \mu_1]\!]\, m_1)\, Q_{\langle \epsilon, \delta \rangle}\, ([\![x \overset{\$}{\leftarrow} \mu_2]\!]\, m_2)$$

where $Q = \{(m_1, m_2) \mid m_1\, x = m_2\, x\}$.

APPENDIX C
PROOF OF THE MAIN THEOREM

Theorem 5 is a corollary of the following lemma:
Lemma 8:

$$\vdash \lceil c \rceil : \Psi \wedge \mathtt{v}_\epsilon = 0 \wedge \mathtt{v}_\delta = 0 \Longrightarrow \Phi \wedge \mathtt{v}_\epsilon \leq \epsilon \wedge \mathtt{v}_\delta \leq \delta$$

implies

$$\forall m_1, m_2.\ m_1 \, \Psi \, m_2 \Rightarrow (\llbracket c \rrbracket\, m_1)\, \Phi_{\langle \epsilon, \delta \rangle}\, (\llbracket c \rrbracket\, m_2)$$

Proof: We first introduce some new notation. For any disjoint memories m_1, m_2 and real values ϵ, δ, $m_1 \oplus_{\epsilon,\delta} m_2$ denotes the memory m such that $m\,x = m_1\,x$ for every $x \in \mathsf{dom}(m_1)$, $m\,x = m_2\,x$ for every $x \in \mathsf{dom}(m_2)$, and $m\,\mathtt{v}_\epsilon = \epsilon$ and $m\,\mathtt{v}_\delta = \delta$. Given a memory relation $R \subseteq \mathcal{M} \times \mathcal{M}$, we let $\hat{R}_{\langle \epsilon, \delta \rangle}$ stand for the set $\{m_1 \oplus_{\epsilon', \delta'} m_2 \mid (m_1, m_2) \in R \wedge \epsilon' \leq \epsilon \wedge \delta' \leq \delta\}$. The proof follows by structural induction on c, proving the following lemma: let $R, Q \subseteq \mathcal{M} \times \mathcal{M}$ be relations on memories, then

$$\begin{array}{l}\left(\forall m.\ m \in \hat{R}_{\epsilon,\delta} \Rightarrow \forall m'.\ m' \in (\llbracket \lceil c \rceil \rrbracket\, m) \Rightarrow m' \in \hat{Q}_{\epsilon',\delta'}\right) \\ \Longrightarrow \\ \forall m_1, m_2.\ (m_1, m_2) \in R \Rightarrow (\llbracket c \rrbracket\, m_1)\, Q_{\langle \epsilon' - \epsilon, \delta' - \delta \rangle}\, (\llbracket c \rrbracket\, m_2)\end{array}$$

Indeed, by setting $\epsilon = 0$ and $\epsilon' = \epsilon$, we get the statement of Lemma 8.

- *Sequential composition:* Let $(m_1, m_2) \in R$. By definition, $m_1 \oplus_{\epsilon,\delta} m_2 \in \hat{R}_{\epsilon,\delta}$. Since $m'\,\mathtt{v}_\epsilon = m''\,\mathtt{v}_\epsilon$ for all $m', m'' \in \llbracket \lceil c_1 \rceil \rrbracket\,(m_1 \oplus_{\epsilon,\delta} m_2)$, then there are ϵ_0, δ_0 and $S \subseteq \mathcal{M} \times \mathcal{M}$ such that $\hat{S}_{\epsilon_0.\delta_0} = \llbracket \lceil c_1 \rceil \rrbracket\, m_1 \oplus_{\epsilon,\delta} m_2$. Also, from the hypotheses, for all $m \in \hat{S}_{\epsilon,\delta}$ we have that $\forall m' \in (\llbracket \lceil c_2 \rceil \rrbracket\, m).\ m' \in \hat{Q}_{\epsilon',\delta'}$. By inductive hypothesis we have thus
  1) $(\llbracket c_1 \rrbracket\, m_1)\, S_{\langle \epsilon_0 - \epsilon, \delta_0 - \delta \rangle}\, (\llbracket c_1 \rrbracket\, m_2)$
  2) for all m', m'' such that $(m', m'') \in S$, we have $(\llbracket c_2 \rrbracket\, m') Q_{\epsilon' - \epsilon_0, \delta' - \delta_0} (\llbracket c_2 \rrbracket\, m'')$

  It follows from Lemma 3 that $(\llbracket c_2 \rrbracket^\star (\llbracket c_1 \rrbracket\, m_1))\, Q_{\langle \epsilon - \epsilon, \delta' - \delta \rangle}\, (\llbracket c_2 \rrbracket^\star (\llbracket c_1 \rrbracket\, m_2))$.
- *While loop:* We start by proving the following auxiliary result:

$$\begin{array}{l}(\forall m.\ m \in \hat{R}_{\epsilon,\delta} \Rightarrow (b_1 = b_2)\, m \wedge \hat{w}_i\, m \neq \bot \wedge \forall m' \in (\hat{w}_i\, m).\ m' \in \hat{Q}_{\epsilon',\delta'}) \\ \Longrightarrow \\ \forall m_1 m_2.\ (m_1, m_2) \in R \Rightarrow (w_i\, m_1)\, Q_{\langle \epsilon' - \epsilon, \delta' - \delta \rangle}\, (w_i\, m_2)\end{array}$$

  The proof follows by natural induction on i. The case $i = 0$ is trivial. For the inductive step, let $m_1, m_2 \in R$. Since $m_1 \oplus_{\epsilon,\delta} m_2 \in \hat{R}_{\epsilon,\delta}$, by hypothesis we have $m_1\, b_1 \Leftrightarrow m_2\, b_2$. We proceed by case analysis on $m_1\, b_1$.
  - In the case $\neg m_1\, b_1$, by definition of $\hat{w}_{i+1}$, $\hat{w}_{i+1}\, m_1 \oplus_{\epsilon,\delta} m_2 = m_1 \oplus_{\epsilon,\delta} m_2$, and thus by hypothesis $m_1 \oplus_{\epsilon,\delta} m_2 \in \hat{Q}_{\epsilon',\delta'}$, which implies $\epsilon = \epsilon'$ and $\delta = \delta'$. By Lemma 4, $\mathbb{1}_{m_1}\, Q_{\langle 0,0 \rangle}\, \mathbb{1}_{m_2}$, which concludes the proof case since we have as well $w_{i+1}\, m_1 = \mathbb{1}_{m_1}$ and $w_{i+1}\, m_2 = \mathbb{1}_{m_2}$.
  - If $m_1\, b_1$ holds, then

$$\hat{w}_{i+1}\, m = \bigcup_{m' \in \llbracket \lceil c \rceil;\, \mathsf{assert}\,(b_1 \Leftrightarrow b_2) \rrbracket} w_i\, m'$$

    Since $b_1 \Leftrightarrow b_2$ is deterministic in $\llbracket \lceil c \rceil \rrbracket\, m$ and $\hat{w}_{i+1}\, m \neq \emptyset$ by hypothesis, then

$$\hat{w}_{i+1}\, m = \bigcup_{m' \in \llbracket \lceil c \rceil \rrbracket} w_i\, m'$$

    By the same reasoning as with sequential composition, there is then S, ϵ_0, and δ_0 such that $\hat{S}_{\epsilon,\delta} = \llbracket \lceil c \rceil \rrbracket\, m_1 \oplus_{\epsilon,\delta} m_2$. Then, by the structural inductive hypothesis we have $(\llbracket c \rrbracket\, m_1)\, S_{\langle \epsilon_0 - \epsilon, \delta_0 - \delta \rangle}\, (\llbracket c \rrbracket\, m_2)$, and by the natural induction hypothesis

$$\forall m_1 m_2.\ (m_1, m_2) \in S \Rightarrow (w_i\, m_1)\, Q_{\langle \epsilon' - \epsilon_0, \delta' - \delta_0 \rangle}\, (w_i\, m_2)$$

    We can conclude from Lemma 3 that

$$(w_{i+1}\, m_1)\, Q_{\langle \epsilon' - \epsilon, \delta' - \delta \rangle}\, (w_{i+1}\, m_2)$$

  It remains to show that the property holds as well when considering the lubs $\bigsqcup \hat{w}_i$ and $\bigsqcup w_i$:

$$\begin{array}{l}(\forall m.\ m \in \hat{R}_{\epsilon,\delta} \Rightarrow (b_1 = b_2)\, m \wedge \forall m' \in (\bigsqcup \hat{w}_i\, m).\ m' \in \hat{Q}_{\epsilon',\delta'}) \\ \Longrightarrow \\ \forall m_1 m_2.\ (m_1, m_2) \in R \Rightarrow (\bigsqcup w_i\, m_1)\, Q_{\langle \epsilon' - \epsilon, \delta' - \delta \rangle}\, (\bigsqcup w_i\, m_2)\end{array}$$

Let m_1 and m_2 such that $(m_1, m_2) \in R$. Since $m_1 \oplus_{\epsilon,\delta} m_2$ then $\forall m' \in (\bigsqcup \hat{w}_i\, m)$. $m' \in \hat{Q}_{\epsilon',\delta'}$. Since we are considering terminating program loops, there exists k such that for all $j \geq k$:

$$\hat{w}_j(m_1 \oplus \epsilon, \delta m_2) \neq \emptyset$$

and furthermore

$$\hat{w}_j(m_1 \oplus \epsilon, \delta m_2) = \bigsqcup \hat{w}_i(m_1 \oplus \epsilon, \delta m_2)$$

From the auxiliary lemma above we have thus

$$(w_j m_1)\, Q_{\langle \epsilon - \epsilon', \delta - \delta' \rangle}\, (w_j m_2)$$

for all $j \geq k$. Since the loop termination condition is deterministic by assumption then it also holds that $w_j m_1 = \bigsqcup_i w_i m_1$ and $w_j m_2 = \bigsqcup_i w_i m_2$ for all $j \geq k$. Then we can conclude:

$$(\bigsqcup_i w_i m_1)\, Q_{\langle \epsilon - \epsilon', \delta - \delta' \rangle}\, (\bigsqcup_i w_i m_2)$$

- *Laplace mechanism:* We consider the case $x \xleftarrow{\$} \mathsf{Lap}_\epsilon(e)$. Let m_1 and m_2 such that $(m_1, m_2) \in R$. Then $m_1 \oplus_{\epsilon_0,\delta_0} m_2 \in \hat{R}_{\epsilon_0,\delta_0}$. From the hypothesis $[\![\lceil c \rceil]\!]\, m \subseteq \hat{Q}_{\epsilon',\delta'}$ and the semantics of the target language, we get

$$\bigcup_{v \in \mathbb{R}} (m_1\, \{v/x\}) \oplus_{\epsilon_0 + |[\![e]\!]\, m_1 - [\![e]\!]\, m_2|\epsilon, \delta_0} (m_2\, \{v/x\}) \subseteq \hat{Q}_{\epsilon',\delta'}$$

From this, we can conclude $\delta' = \delta_0$, $\epsilon' = \epsilon_0 + |[\![e]\!]\, m_1 - [\![e]\!]\, m_2|\epsilon$ and

$$Q \supseteq \{(m_1, m_2) \mid \exists v_1, v_2.\ (m_1\, \{v_1/x\}, m_2\, \{v_2/x\}) \in R\} \cap \{(m_1, m_2) \mid m_1\, x = m_2\, x\}$$

Since the first term in the intersection above is preserved by any assignment to the x variable, by Lemma 6 it is enough to consider the case $Q = \{(m_1, m_2) \mid m_1\, x = m_2\, x\}$, and prove $([\![c]\!]\, m_1)\, Q_{\langle |[\![e]\!]\, m_1 - [\![e]\!]\, m_2|\epsilon, 0 \rangle}\, ([\![c]\!]\, m_2)$. To verify this, by Lemma 7, it is sufficient to show that

$$\Delta_{|[\![e]\!]\, m_1 - [\![e]\!]\, m_2|\epsilon}(\mathsf{Lap}_\epsilon([\![e]\!]\, m_1), \mathsf{Lap}_\epsilon([\![e]\!]\, m_2)) \leq 0$$

We need to show that for every r we have

$$\mathsf{Lap}_\epsilon([\![e]\!]\, m_1)\, r - \exp(|[\![e]\!]\, m_1 - [\![e]\!]\, m_2|\epsilon)\mathsf{Lap}_\epsilon([\![e]\!]\, m_2)\, r \leq 0$$

Then, it is enough to prove:

$$\left(\frac{\exp\left(-\frac{\epsilon|r - [\![e]\!]\, m_1|}{2}\right)}{\sum_{r'} \exp\left(-\frac{\epsilon|r' - [\![e]\!]\, m_1|}{2}\right)} \right) - \exp(|[\![e]\!]\, m_1 - [\![e]\!]\, m_2|\epsilon) \left(\frac{\exp\left(-\frac{\epsilon|r - [\![e]\!]\, m_2|}{2}\right)}{\sum_{r'} \exp\left(-\frac{\epsilon|r' - [\![e]\!]\, m_2|}{2}\right)} \right) \leq 0$$

This is equivalent to prove

$$\frac{\exp\left(-\frac{\epsilon|r - [\![e]\!]\, m_1|}{2}\right) \cdot \sum_{r' \in \mathcal{R}} \exp\left(-\frac{\epsilon|r' - [\![e]\!]\, m_2|}{2}\right)}{\exp\left(-\frac{\epsilon|r - [\![e]\!]\, m_2|}{2}\right) \cdot \sum_{r' \in \mathcal{R}} \exp\left(-\frac{\epsilon|r' - [\![e]\!]\, m_1|}{2}\right)} \leq \exp(|[\![e]\!]\, m_1 - [\![e]\!]\, m_2|\epsilon)$$

The first term can be bound by

$$\exp\left(\frac{\epsilon|[\![e]\!]\, m_2 - [\![e]\!]\, m_1|}{2}\right) \cdot \frac{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\epsilon|r' - [\![e]\!]\, m_2|}{2}\right)}{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\epsilon|r' - [\![e]\!]\, m_1|}{2}\right)}$$

For every $r' \in \mathcal{R}$, we know $|r' - [\![e]\!]\, m_2| \geq |r' - [\![e]\!]\, m_1| - |[\![e]\!]\, m_2 - [\![e]\!]\, m_1|$. So, the above can be bound by

$$\exp\left(\frac{\epsilon|[\![e]\!]\, m_2 - [\![e]\!]\, m_1|}{2}\right) \cdot \frac{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\epsilon(|r' - [\![e]\!]\, m_1| - |[\![e]\!]\, m_2 - [\![e]\!]\, m_1|)}{2}\right)}{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\epsilon|r' - [\![e]\!]\, m_1|}{2}\right)}$$

that is equivalent to

$$\exp\left(\frac{\epsilon|[\![e]\!]\, m_2 - [\![e]\!]\, m_1|}{2}\right) \cdot \exp\left(\frac{\epsilon|[\![e]\!]\, m_2 - [\![e]\!]\, m_1|}{2}\right) \cdot \frac{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\epsilon|r' - [\![e]\!]\, m_1|}{2}\right)}{\sum_{r' \in \mathcal{R}} \exp\left(-\frac{\epsilon|r' - [\![e]\!]\, m_1|}{2}\right)}$$

and simplifying

$$\mathsf{exp}\left(\frac{\epsilon|[\![e]\!]\,m_2 - [\![e]\!]\,m_1|}{2}\right)\cdot\mathsf{exp}\left(\frac{\epsilon|[\![e]\!]\,m_2 - [\![e]\!]\,m_1|}{2}\right)$$

that is what we need.

- *Exponential mechanism:* Following a similar reasoning to the Laplace mechanism case, we need to prove that for every r we have

$$\mathsf{Exp}_\epsilon([\![s]\!]\,m_1, [\![e]\!]\,m_1)\,r - \mathsf{exp}(\epsilon_1-\epsilon_0)\mathsf{Exp}_\epsilon([\![s]\!]\,m_2, [\![e]\!]\,m_2)\,r \leq 0$$

where $\mathsf{Exp}_\epsilon(s,x)$ stands for the distribution

$$\lambda r.\frac{\mathsf{exp}\left(\frac{\epsilon s(x,r)}{2}\right)}{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon s(x,r')}{2}\right)}$$

By Lemma 5 and the fact that $[\![s]\!]m_1 = [\![s]\!]m_2 = \hat{s}$ it is then enough to prove:

$$\frac{\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_1,r)}{2}\right)}{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_1,r')}{2}\right)} - \mathsf{exp}(\epsilon_1-\epsilon_0)\frac{\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_2,r)}{2}\right)}{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_2,r')}{2}\right)}) \leq 0$$

This is equivalent to prove

$$\frac{\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_1,r)}{2}\right)\cdot\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_2,r')}{2}\right)}{\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_2,r)}{2}\right)\cdot\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_1,r')}{2}\right)} \leq \mathsf{exp}(\epsilon_1-\epsilon_0)$$

Continuing we have

$$\mathsf{exp}\left(\frac{\epsilon(\hat{s}([\![e]\!]m_1,r)-\hat{s}([\![e]\!]m_2,r))}{2}\right)\cdot\frac{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_2,r')}{2}\right)}{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_1,r')}{2}\right)} \leq \mathsf{exp}(\epsilon_1-\epsilon_0)$$

Using the fact that $\max_{r\in\mathcal{R}}|\hat{s}(e_1,r)-\hat{s}(e_2,r)|\epsilon \leq \epsilon_1-\epsilon_0$ we have:

$$\mathsf{exp}\left(\frac{\epsilon_1-\epsilon_0}{2}\right)\cdot\frac{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_2,r')}{2}\right)}{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_1,r')}{2}\right)} \leq \mathsf{exp}(\epsilon_1-\epsilon_0)$$

Using the same fact we also know that for every $r'\in\mathcal{R}$ we have $\hat{s}(e_2,r) \leq \frac{\epsilon_1-\epsilon_0}{\epsilon} + \hat{s}(e_1,r)$. So,we have:

$$\mathsf{exp}\left(\frac{\epsilon_1-\epsilon_0}{2}\right)\cdot\frac{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{(\epsilon_1-\epsilon_0)+\epsilon\hat{s}([\![e]\!]m_1,r')}{2}\right)}{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_1,r')}{2}\right)} \leq \mathsf{exp}(\epsilon_1-\epsilon_0)$$

that is equivalent to

$$\mathsf{exp}\left(\frac{\epsilon_1-\epsilon_0}{2}\right)\cdot\mathsf{exp}\left(\frac{\epsilon_1-\epsilon_0}{2}\right)\cdot\frac{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_1,r')}{2}\right)}{\sum_{r'\in\mathcal{R}}\mathsf{exp}\left(\frac{\epsilon\hat{s}([\![e]\!]m_1,r')}{2}\right)} \leq \mathsf{exp}(\epsilon_1-\epsilon_0)$$

and simplifying

$$\mathsf{exp}\left(\frac{\epsilon_1-\epsilon_0}{2}\right)\cdot\mathsf{exp}\left(\frac{\epsilon_1-\epsilon_0}{2}\right) \leq \mathsf{exp}(\epsilon_1-\epsilon_0)$$

■

Lemma 9 (Proof of the accuracy specification):

$$\forall m_1, m_2.\ [\![e]\!]\,m_1 = [\![e]\!]\,m_2 \Rightarrow ([\![x \leftarrow \mathsf{Lap}_\epsilon(e)]\!]\,m_1)\,Q_{\langle 0,\delta\rangle}\,([\![x \leftarrow \mathsf{Lap}_\epsilon(e)]\!]\,m_2)$$

where

$$Q \doteq \{(m_1,m_2) \mid m_1\,x = m_2\,x \wedge |m_1\,x - [\![e]\!]\,m_1| \leq \log(2/\delta)/(2\epsilon)\}$$

Proof: We need to prove:

$$([\![x \xleftarrow{\$} \mathsf{Lap}_\epsilon(e)]\!]\,m_1)\,Q_{\langle 0,\delta\rangle}\,([\![x \xleftarrow{\$} \mathsf{Lap}_\epsilon(e)]\!]\,m_2)$$

By the assumption $e_1 = e_2$ we have that $[\![x \xleftarrow{\$} \mathsf{Lap}_\epsilon(e)]\!]\, m_1$ and $[\![x \xleftarrow{\$} \mathsf{Lap}_\epsilon(e)]\!]\, m_2$ are the same distribution $\hat{\mu}$. Now, consider the set $S = \{z : \mathbb{R} \mid |z - [\![e_1]\!]\, m_1| < \log(2/\delta)/(2\epsilon)\}$ and the distribution $\mu \in \mathcal{D}(\mathbb{R} \times \mathbb{R})$, parametrized on S defined as:

$$\mu(z_1, z_2) := \begin{cases} \hat{\mu}\, z_1 & \text{if } z_1 = z_2 \wedge z_1 \in S \\ 0 & \text{otherwise.} \end{cases}$$

Notice that by definition of $\hat{\mu}$ we have $\pi_1\mu \leq [\![x \xleftarrow{\$} \mathsf{Lap}_\epsilon(e)]\!] m_1$ and $\pi_2\mu \leq [\![x \xleftarrow{\$} \mathsf{Lap}_\epsilon(e)]\!] m_2$. Moreover, by definition of S we also have that for every m, $\mu\, m \neq 0 \Rightarrow \Phi\, m$. Since clearly $\pi_1\mu = \pi_2\mu$, the only thing left to prove is that $\Delta_0(\hat{\mu}, \pi_1\mu) \leq \delta$. This means that we need to prove

$$\max_{R \subseteq \mathbb{R}} \{\hat{\mu}\, R - \pi_1\mu\, R\} \leq \delta$$

It is easy to see that on values in $\mathbb{R} \cap S$ the two distribution coincide. So we can instead consider

$$\max_{R \subseteq (\mathbb{R}/S)} \{\hat{\mu}\, R - \pi_1\mu\, R\} \leq \delta$$

Now, notice that for every $R \subseteq (\mathbb{R}/S)$ we have $\pi_1\mu = 0$, so we can just consider

$$\max_{R \subseteq (\mathbb{R}/S)} \{\hat{\mu}\, R\} \leq \delta$$

and since by definition $\hat{\mu}\, R = \sum_{a \in R} \hat{\mu}\, a$ where every value is non-negative, we can just consider $\hat{\mu}\,(\mathbb{R}/S) \leq \delta$. Now, recall that S corresponds to the interval:

$$[-(\log(2/\delta)/(2\epsilon)) + [\![e]\!]\, m_1, [\![e]\!]\, m_1 + (\log(1/\delta)/\epsilon)]$$

and that $\hat{\mu} = [\![x \xleftarrow{\$} \mathsf{Lap}_\epsilon(e)]\!]\, m_1$. So, we can apply a tail bound on the Laplace distribution:

$$\{\hat{\mu}\, z \mid z \in (\mathbb{R}/S)\} = \{\hat{\mu}\, z \mid |z - [\![e]\!]\, m_1| \geq \log(2/\delta)/(2\epsilon)\} < \delta.$$

and conclude

$$\hat{\mu}\,(\mathbb{R}/S) \leq \delta$$

that is what we need. ■

Lemma 10 (Tail bound for the discrete version of Laplace): Let x be drawn from the discrete version of the Laplace distribution with mean 0 and parameter $b > 0$, i.e., with probability

$$L_b(x) = \frac{\mathsf{exp}\left(-\frac{|x|}{2b}\right)}{\sum_{z \in \mathbb{Z}} \mathsf{exp}\left(-\frac{|z|}{2b}\right)}.$$

Then, for $T \in \mathbb{N}$ and $T > 0$:

$$\Pr\left[x : |L_b(x)| > T\right] \leq 2\,\mathsf{exp}\left(-\frac{T}{2b}\right).$$

In particular, if $b = 1/\epsilon$ (like in $\mathsf{Lap}_\epsilon(x)$) and $T = \log(2/\delta)/(2\epsilon)$, we have Lemma 1.

Proof: We have

$$\begin{aligned}
\Pr\left[x : |L_b(x)| > T\right] &= \Pr\left[x : \left|\frac{\mathsf{exp}\left(-\frac{|x|}{2b}\right)}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)}\right| > T\right] \\
&= \Pr\left[x : \frac{\mathsf{exp}\left(-\frac{|x|}{2b}\right)}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)} > T\right] + \Pr\left[x : \frac{\mathsf{exp}\left(-\frac{|x|}{2b}\right)}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)} < -T\right] \\
&= \sum_{x=T}^{\infty}\frac{\mathsf{exp}\left(-\frac{|x|}{2b}\right)}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)} + \sum_{-T}^{-\infty}\frac{\mathsf{exp}\left(-\frac{|x|}{2b}\right)}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)} \\
&= \frac{\sum_{x=T}^{\infty}\mathsf{exp}\left(-\frac{x}{2b}\right) + \sum_{-T}^{-\infty}\mathsf{exp}\left(\frac{x}{2b}\right)}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)} \\
&= \frac{\sum_{x=0}^{\infty}\mathsf{exp}\left(-\frac{x+T}{2b}\right) + \sum_{x=0}^{\infty}\mathsf{exp}\left(\frac{-x-T}{2b}\right)}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)} \\
&= \frac{\mathsf{exp}\left(-\frac{T}{2b}\right)\ \sum_{x=0}^{\infty}\mathsf{exp}\left(-\frac{x}{2b}\right) + \mathsf{exp}\left(-\frac{T}{2b}\right)\ \left(1 + \sum_{x=1}^{\infty}\mathsf{exp}\left(-\frac{x}{2b}\right)\right)}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)} \\
&= \mathsf{exp}\left(-\frac{T}{2b}\right)\ \frac{\left(\sum_{x=0}^{\infty}\mathsf{exp}\left(-\frac{x}{2b}\right) + 1 + \sum_{x=1}^{\infty}\mathsf{exp}\left(-\frac{x}{2b}\right)\right)}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)} \\
&= \mathsf{exp}\left(-\frac{T}{2b}\right)\ \left(\frac{1}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)} + \frac{\left(\sum_{x=0}^{\infty}\mathsf{exp}\left(-\frac{x}{2b}\right) + \sum_{x=1}^{\infty}\mathsf{exp}\left(-\frac{x}{2b}\right)\right)}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)}\right) \\
&= \mathsf{exp}\left(-\frac{T}{2b}\right)\ \left(\frac{1}{\sum_{z\in\mathbb{Z}}\mathsf{exp}\left(-\frac{|z|}{2b}\right)} + 1\right) \\
&\le 2\,\mathsf{exp}\left(-\frac{T}{2b}\right)
\end{aligned}$$

■

APPENDIX D
VERIFICATION OF VERTEX COVER

The extended logic used to prove the vertex cover in apRHL features a more precise rule for while loops, that allows the privacy budget to vary at each iteration

$$\frac{\begin{array}{c}\Theta \implies b_1 \equiv b_2 \wedge i_1 = i_2 \qquad \Theta \wedge n \le i_1 \implies \neg b_1 \\ \vdash c_1 \sim_{\langle \epsilon_j, \delta_j\rangle} c_2 : \Theta \wedge b_1 \wedge i_1 = j \Longrightarrow \Theta \wedge i_1 = j{+}1\end{array}}{\vdash \mathsf{while}\ b_1\ \mathsf{do}\ c_1 \sim_{\langle \sum_{i=0}^{n-1}\epsilon_i, \sum_{i=0}^{n-1}\delta_i\rangle} \mathsf{while}\ b_2\ \mathsf{do}\ c_2 : \Theta \wedge i_1 = 0 \Longrightarrow \Theta \wedge \neg b_1}$$

and a code motion rule that allows to swap statements c_1 and c_2 provided they satisfy some independence condition:

$$\vdash c_1; c_2 \sim_{\langle 0,0\rangle} c_2; c_1 : \forall x \in X. x_1 = x_2 \Longrightarrow \forall x \in X. x_1 = x_2$$

In addition, the extended logic features a transitivity rule that allows to compose apRHL judgments. These rules can be readily encoded in our setting, provided we allow for more general forms of products as considered in [4], [5].

However, the extended logic also considers a probabilistic programming language with assert statements, and ad hoc rules for random assignments and while loops:

$$\frac{\begin{array}{l}\Theta \implies b_1 \equiv b_2 \wedge P_1 \equiv P_2 \\ \vdash c_1;\ \mathsf{assert}\,(P_1) \sim_{\langle \epsilon, \delta\rangle} c_2;\ \mathsf{assert}\,(P_2) : \Theta \wedge b_1 \wedge \neg P_1 \Longrightarrow \Theta \\ \vdash c_1 \sim_{\langle 0,0\rangle} c_2 : \Theta \wedge b_1 \Longrightarrow \Theta \\ \vdash c_1 \sim_{\langle 0,0\rangle} c_2 : \Theta \wedge b_1 \wedge P_1 \Longrightarrow \Theta \wedge P_1\end{array}}{\vdash \mathsf{while}\ b_1\ \mathsf{do}\ c_1 \sim_{\langle \epsilon, \delta\rangle} \mathsf{while}\ b_2\ \mathsf{do}\ c_2 : \Theta \Longrightarrow \Theta \wedge \neg b_1}$$

These rules are not captured by our approach.

For comparison, we briefly describe the proof in apRHL and the relational specifications of choose that are required for completing the proof. For the first case, the apRHL proof uses the first generalized loop rule, and the following property of choose:

$$\vdash v_1 \leftarrow \mathsf{choose}_{\epsilon,n}(g_1) \sim_{\left\langle \epsilon/\left(2\sqrt{n}\sqrt{|g_1|}\right),0\right\rangle} v_2 \leftarrow \mathsf{choose}_{\epsilon,n}(g_2) : g_1 \cup \{\langle u_1, t_1\rangle\} = g_2 \Longrightarrow v_1 = v_2$$

In the second case, the apRHL uses the second generalized loop rule, and the following properties of choose:

$$\vdash \left(\begin{array}{c} v_1 \leftarrow \mathsf{choose}_{\epsilon,n}(g_1); \\ \mathsf{assert}\,(v_1 \neq u_1 \wedge v_1 \neq t_1) \end{array}\right) \sim_{\langle 0,0\rangle} \left(\begin{array}{c} v_2 \leftarrow \mathsf{choose}_{\epsilon,n}(g_2); \\ \mathsf{assert}\,(v_2 \neq u_2 \wedge v_2 \neq t_2) \end{array}\right) : g_1 = g_2 \cup \{\langle u_2, t_2\rangle\} \Longrightarrow v_1 = v_2,$$

$$\vdash \left(\begin{array}{c} v_1 \leftarrow \mathsf{choose}_{\epsilon,n}(g_1); \\ \mathsf{assert}\,(v_1 = u_1 \vee v_1 = t_1) \end{array}\right) \sim_{\langle \frac{\epsilon}{4},0\rangle} \left(\begin{array}{c} v_2 \leftarrow \mathsf{choose}_{\epsilon,n}(g_2); \\ \mathsf{assert}\,(v_2 = qu_2 \vee v_2 t_2) \end{array}\right) : g_1 = g_2 \cup \{\langle u_2, t_2\rangle\} \Longrightarrow v_1 = v_2.$$