



**HAL**  
open science

# Session Types for the Concurrent Composition of Interactive Differential Privacy

Victor Sannier, Patrick Baillot, Marco Gaboardi

► **To cite this version:**

Victor Sannier, Patrick Baillot, Marco Gaboardi. Session Types for the Concurrent Composition of Interactive Differential Privacy. 2024. hal-04719333

**HAL Id: hal-04719333**

**<https://hal.science/hal-04719333v1>**

Preprint submitted on 3 Oct 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Session Types for the Concurrent Composition of Interactive Differential Privacy

Victor Sannier<sup>1</sup>, Patrick Baillot<sup>1</sup>, and Marco Gaboardi<sup>2</sup>

<sup>1</sup>Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRISTAL, F-59 000 Lille, France

<sup>2</sup>Department of Computer Science, Boston University, USA

October 2, 2024

## Abstract

Differential privacy (DP) is a statistical definition of privacy which ensures that the outcome of a computation by an analyst only depends in a negligible way on the presence of a single record in the dataset. This framework has been extended first to the interactive setting where the analyst can ask an adaptive sequence of queries, and then to the concurrent interactive setting where the adaptive queries can be performed concurrently on the dataset. A key result of such frameworks is that of composition theorems, which allow data curators to combine several algorithms to obtain a new algorithm that continues to satisfy DP, with certain privacy parameters. Obtaining composition theorems in the concurrent interactive framework and for advanced notions of DP is delicate and some results have been obtained recently [1, 2]. On the other hand a variety of tools have been proposed for certifying that some given analysis are DP. Among them, the typing approach embodied by the Fuzz language consists in using a functional programming language and a type system ensuring that well-typed programs can automatically be rendered differentially private. However this setting does not allow to represent concurrent interactive systems. We thus propose to extend it by using as language a process calculus similar to the  $\pi$ -calculus, and we define a trace semantics for it which allows to formulate the DP property as an approximation trace equivalence.

Moreover, we introduce a type system in the approach of session types and prove a soundness result stating that if a system of processes is well-typed, then it is differentially private.

## 1 Introduction

### 1.1 Verification of Differential Privacy

Differential privacy (DP) is a wide-spread and popular notion in the area of data protection [3, 4]. Its advantages over some other privacy notions are that it allows to obtain mathematically robust results and that it enjoys in its various settings a key property of composability: a data curator can combine various DP algorithms in order to obtain a new DP algorithm. The literature has proposed and explored many notions of DP which account for various refinement levels of privacy such as for instance pure or  $(\epsilon, 0)$ -DP,  $(\epsilon, \delta)$ -DP, Renyi-DP [5],  $f$ -DP [6] among others. Depending on the variant of DP and on the setting considered, the parameters obtained for the resulting algorithm ensure a privacy bound that can vary.

However, verifying in practice that certain specific composed programs are differentially private can be tedious and subtle [7]. See Gaboardi et al. [8], Bun et al. [9] for results on the high complexity of this verification. For this reason, several tools based on pro-

programming languages theory have been proposed for assisting a programmer in checking whether a given program is differentially private or not [10]. Among them, the Fuzz approach [11] involves using a functional language to write the analyst’s queries. It offers a type system that statically ensures that a well-typed query can automatically be rendered differentially private.

This line of work in type-based differential privacy (DP) verification has continued and expanded in various directions. For instance: DFuzz [12] advanced this area by offering more detailed sensitivity analyses through the use of linear dependent types. Some works have focused on dealing with  $(\epsilon, \delta)$ -DP [13, 14, 15] and on considering the Euclidean metrics for vectors and matrices [14]. Others [16, 17] have explored how to handle various metrics within type constructions.

Still, this one-shot setting where the DP property is expressed for a single query is only a particular situation and cannot account for some increasingly common scenarios where the data analyst interacts with the dataset in a sequential adaptive way and the of primitives such as the Sparse Vector Technique (SVT) [18, 19], and the Private Multiplicative Weights [20]. This leads to the study of interactive mechanisms to obtain DP properties for such algorithms.

Moreover in some cases analysts might want to perform multiple adaptive analyses on the same dataset concurrently by interleaving their queries to several differentially private mechanisms (see Figure 1, reproduced from Vadhan and Zhang [21, Figure 3]). They may, therefore, correlate their queries based on the answers received from other mechanisms. This setting can be seen as a type of multiparty interaction for which several composition theorems have been proved in recent years for various differential privacy definitions [1, 2].

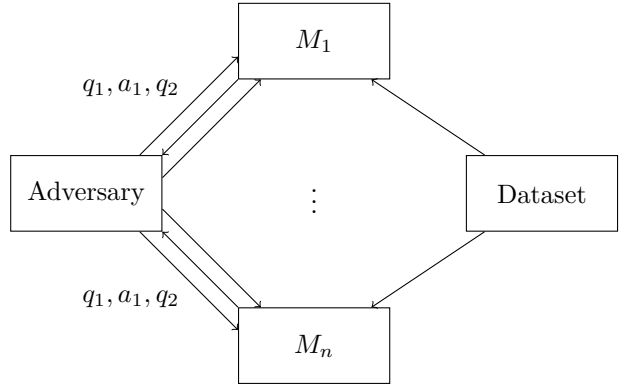


Figure 1: Concurrent composition of interactive mechanisms, where the adversary can arbitrarily interleave its queries

## 1.2 A Type System for the Concurrent Composition of Interactive Differential Privacy

The fact that verifying whether an interaction in a specific multiparty system satisfies differential privacy is even more challenging than in the original single-query settings raises the question of how one could extend the typing approach, as illustrated by the Fuzz language, to this more complex framework.

The first ingredient needed is a language to represent distributed systems composed of adversaries and mechanisms, in a mathematical way. We propose to turn towards process calculi derived from the  $\pi$ -calculus [22, 23]. They have been successfully used in the literature [24] to represent interaction protocols and to offer a solid basis on which to reason on the properties of such protocols.

The second ingredient required is a theory to reason on the privacy information in this setting. Indeed as the communication is not any more a simple query-answer, but consists in two-way flows of information between an analyst/adversary and each mechanism, one needs a representation of this communication and a suitable definition of DP. We propose for this an operational semantics based on fully probabilistic labelled transition systems for representing the behaviour of the system, and to define DP in

this setting by drawing inspiration from the ideas introduced in Vadhan and Wang [1]. We will need in particular to formalise what each processes *sees* from the computation and how to quantitatively compare computations.

Finally the third ingredient is a type system for this  $\pi$ -calculus, in which one could integrate the parameters needed for the DP analysis. To address this, we will introduce a type system in the format of session types [24], an approach that has been extensively investigated in the literature, providing a robust and versatile framework.

Once these three ingredients have been introduced, our goal will be to prove a soundness theorem in the following form: if a system of distributed mechanisms is well-typed, then for any adversarial analyst the global system composed of the analyst and the mechanisms satisfies a DP property, expressed with some parameters given by the types. In particular, in our type system, the role of composition theorems will be fulfilled by the soundness of the typing rule for the parallel composition of two (or more, by iterating) mechanisms.

### 1.3 Contributions

In this article,

- we introduce a variant of the  $\pi$ -calculus with sessions that includes constructs for finite replication and random number generation, and provide a probabilistic operational semantics for it;
- we provide typing rules that, in addition to ensuring the usual safety properties, also track the privacy parameters of the processes: specifically, we introduce rules for the concurrent composition and the parallel composition of two interactive processes;
- we offer a syntactical definition of the view of a process to formally define interactive differential privacy, and we prove that our typing rules are sound with respect to this definition; and lastly
- we demonstrate how to write various programs such as an implementation of the private Guess-and-Check algorithm in our process calculus and

statically prove that it guarantees interactive differential privacy.

## 2 Preliminaries

### 2.1 Differential Privacy

Differential privacy requires that the outcome of a computation is approximately the same when a single record, typically associated with a given individual, is added to or removed from the input data [3].

**Definition 2.1** ([4, Definition 2.4]) *A randomised algorithm  $\mathcal{M}$  is  $(\epsilon, \delta)$ -differentially private if for all  $X \subseteq \text{Range}(\mathcal{M})$  and for all adjacent inputs  $x$  and  $y$ , we have*

$$\Pr[\mathcal{M}(x) \in X] \leq e^\epsilon \Pr[\mathcal{M}(y) \in X] + \delta. \quad (1)$$

**Remark 2.1** *Each application domain may utilise its own specific notions of input and adjacency. However, the most common case is that two databases are considered adjacent if one can transition from one to the other by simply adding or removing a record.*

In practice, as mentioned in the introduction, proving that a given algorithm is differentially private is a complex task. This is why the community has focused on developing automatic procedures to transform an algorithm into a differentially private (DP) algorithm with known privacy parameters, typically through the addition of well-calibrated noise to its result. To accomplish this, it is necessary to understand how the algorithm in question responds to slight modifications in its inputs, i.e., its sensitivity, which can be intuitively regarded as an upper bound on its slope at any point.

**Definition 2.2** *A function  $f$  between two metric spaces  $(X, d_X)$  and  $(Y, d_Y)$  is said to be  $s$ -sensitive if, for all inputs  $x$  and  $x'$  in  $X$ , we have*

$$d_Y(f(x), f(x')) \leq s \cdot d_X(x, x'). \quad (2)$$

**Theorem 2.1** ([4, Theorem 3.6]) *For all algorithms  $f$  mapping to  $\mathbf{R}^n$  that are  $s$ -sensitive with respect to the  $L^1$  metric, the randomised algorithm*

$$x \mapsto (f_1(x) + Y_1, \dots, f_n(x) + Y_n) \quad (3)$$

where  $Y_i \sim \text{Lap}_{s/\epsilon}$  for all  $i$  is  $\epsilon$ -differentially private.

**Remark 2.2** See Ghosh et al. [25] for a discrete version of this mechanism, using integers rather than real numbers.

Further automation in verifying differential privacy can involve statically analysing the sensitivity of a program, notably through the use of dedicated type systems.

In particular, Reed and Pierce [11] introduced Fuzz, a typed functional programming language where types are interpreted by metric spaces [26], and whose type constructors are derived from linear logic [27]. Thus,  $A \otimes B$  is interpreted as the product space  $\llbracket A \rrbracket \otimes \llbracket B \rrbracket$  endowed with the  $L^1$ -metric (that is,  $d_{A \otimes B}((a, b), (a', b')) = d_A(a, a') + d_B(b, b')$ ),  $\llbracket A \multimap B \rrbracket$  is the space of all 1-sensitive functions from  $\llbracket A \rrbracket$  to  $\llbracket B \rrbracket$ , and for all sensitivities  $s$ ,  $\llbracket !_s A \rrbracket$  is the space  $\llbracket A \rrbracket$  but endowed with the metric  $d_{!_s A} = s \cdot d_A$ . In addition to this denotational semantics, Fuzz is also equipped with a big-step operational semantics [11, Section 2.6], whose evaluation relation we write as  $\downarrow$ .

Typing judgements are of the form  $[x_1 : A_1]_{s_1}, \dots, [x_n : A_n]_{s_n} \vdash e : A$  and are sound when  $\llbracket e \rrbracket$  is a 1-sensitive function from the product space  $!_{s_1} \llbracket A_1 \rrbracket \otimes \dots \otimes !_{s_n} \llbracket A_n \rrbracket$  endowed with the  $L_1$  distance to the metric space  $\llbracket A \rrbracket$ . They may also read, “For all  $i$ ,  $\llbracket e \rrbracket$  is  $s_i$ -sensitive in the variable  $x_i$  of type  $A_i$ .” Let us provide two examples of deduction rules of such judgements: the introduction and elimination of the tensor product.

$$\frac{\Gamma \vdash a : A \quad \Delta \vdash b : B}{\Gamma + \Delta \vdash (a, b) : A \otimes B} \otimes I$$

$$\frac{\Delta \vdash e : A \otimes B \quad \Gamma, [x : A]_s, [y : B]_s \vdash c : C}{\Gamma + s\Delta \vdash \text{let } (x, y) = c \text{ in } c : C} \otimes E$$

where the addition of two typing contexts is their union, with pointwise addition of the sensitivity values, and  $s\Delta$  is the context obtained by multiplying each sensitivity value in  $\Delta$  by  $s$ .

Finally, a monadic constructor  $\circ$  parametrised by a fixed  $\epsilon$  ensures that if  $\vdash f : A \multimap B$ , then  $\llbracket f \rrbracket$  is  $\epsilon$ -differentially private from  $\llbracket A \rrbracket$  to  $\llbracket B \rrbracket$  meaning that

we have performed the noise addition step within the language itself.

Fuzz is the expression language we will consider in the following, rather than merely a simply-typed  $\lambda$ -calculus. The sensitivity analysis will be crucial, for example, in the typing rule [T-NSEND] (see Figure 3), and in the Guess-and-Check example (see Section 6.3).

## 2.2 Interactive Differential Privacy

However, the framework we have considered so far remains quite limited, as the communication is terminated once a request has been sent and a result obtained. As a consequence, it is not possible for the server to correlate the noise it adds to the results sent to the analyst in order to reduce its amplitude without compromising privacy guarantees [18, 19]. For this reason, a notion of differential privacy has been introduced for interactive protocols.

Let us begin by recalling the definition provided by Vadhan and Wang. An interactive protocol is a pair of mutually recursive functions, referred to as *parties*, and the view of a party is understood as “everything it sees during the execution” [1, Definition 1.6]. We write  $\text{View}(A \parallel M)$  for the view of  $A$  when interacting with  $M$ , and  $\text{View}(M \parallel A)$  for the view of  $M$  when interacting with  $A$ .

**Definition 2.3** ([1, Definition 1.7]) *A randomised interactive algorithm  $\mathcal{M}$  is  $(\epsilon, \delta)$ -differentially private if for all adversaries  $\mathcal{A}$ , for all  $X \subseteq \text{Range}(\text{View}(\mathcal{A} \parallel \mathcal{M}(\cdot)))$  and for all adjacent inputs  $x$  and  $y$ , we have*

$$\Pr[\text{View}(\mathcal{A} \parallel \mathcal{M}(x)) \in X] \leq e^\epsilon \Pr[\text{View}(\mathcal{A} \parallel \mathcal{M}(y)) \in X] + \delta. \quad (4)$$

**Remark 2.3** *Note that we will use the term “adversary” throughout this paper, as we do not know the intentions of  $\mathcal{A}$ . However, in many cases, it will actually refer to a data analyst.*

There are two concepts of composition for interactive differential privacy. The first is parallel composition, where two processes work on disjoint subsets

of the input domain. Therefore, the combined privacy guarantee only depends on the worst guarantee of each mechanism and not on their sum.

**Theorem 2.2** ([28, Theorem 4]) *Let  $M_i$  be mechanisms, each providing  $\epsilon$ -differential privacy. Let  $D_i$  be arbitrary disjoint subsets of the input domain  $D$ . The sequence of  $M_i(X \cap D_i)$  provides  $\epsilon$ -differential privacy.*

The second is concurrent composition, where two processes operate on the same data, potentially interleaving their message sending and receiving. We will only present one result from the literature as an illustration, which is for  $(\epsilon, \delta)$ -differential privacy, where the privacy parameters for all mechanisms are equal.

**Theorem 2.3** ([1, Theorem 1.8]) *If interactive mechanisms  $M_1, \dots, M_n$  are each  $(\epsilon, \delta)$ -differentially private, then their concurrent composition is  $(k \cdot \epsilon, (e^{k\epsilon} - 1)/(e^\epsilon - 1) \cdot \delta)$ -differentially private.*

Accordingly, we will later introduce two typing rules for composition, [T-CONC] and [T-PAR], as shown in Figure 3.

### 2.3 Process Calculi and Session Types

In order to write interactive programs, we shall use a process calculus, specifically the  $\pi$ -calculus [22, 23], rather than functional languages like the  $\lambda$ -calculus. The core concept of the  $\pi$ -calculus is that of processes communicating over channels, enabling two-way interaction.

The syntax of the  $\pi$ -calculus we will use is defined on Figure 2. The simplest process, denoted as  $0$ , is the trivial process that performs no actions. To create a session, i.e., to open a communication channel between two processes, the syntax is  $\bar{a}(k) . P$  for session requests and  $a(k) . P$  for session acceptance. A process can then send an expression  $e$  over a channel  $k$ , and proceed as  $P$ , which is written as  $k![e] . P$ . Conversely, a process can receive an expression over a channel  $k$  into a variable  $x$  and proceed as  $P$ , denoted as  $k?(x) . P$ . The  $\pi$ -calculus also includes control structures such as conditional expressions, **if**  $e$  **then**  $P$  **else**  $Q$ , which choose to continue

as process  $P$  or  $Q$  based on the evaluation of  $e$ . Additionally, it supports branching and selection mechanisms, where a process can offer or choose amongst several labelled branches with  $s \triangleright \{l_i : P_i \mid i \in I\}$  for label branching and  $s \triangleleft l . P$  for label selection. Names and channels can be hidden within a process using  $(\nu a)P$  for name hiding and  $(\nu k)P$  for channel hiding.

Several type systems have been introduced for this language, but we will focus on session types (see Dezani-Ciancaglini and De'Liguoro [29] for an overview), a type system that ensures communication over a given channel adheres to a specified sequence of actions, thus preventing mismatches and communication errors. The grammar of session types is given in Figure 2. Specifically, the type  $?A . \alpha$  reads as “the session expects a value of type  $A$  and then behaves as a session with type  $\alpha$ ” (where  $A$  ranges over Fuzz types). Conversely,  $!A . \alpha$  reads as “the session sends a value of type  $A$  and then behaves as a session with type  $\alpha$ .” Additionally,  $\&_i \{l_i : \alpha_i\}$  and  $\oplus_i \{l_i : \alpha_i\}$  are session types for branching and selection, respectively.

The syntax we adopted here for the  $\pi$ -calculus with sessions is essentially the one of Honda et al. in their seminal paper [24], with some minor modifications in notation and naming influenced by a more recent paper [30] by the same lead author [30].

**Remark 2.4** *We made two omissions from the usual syntax of the  $\pi$ -calculus with session types. First, we chose not to include constructs for sending and receiving channels, that is for process delegation. While such constructs can enhance modularity, facilitate exception handling, and increase concurrency [24, Section 4.3], they are not strictly necessary and would complicate the presentation. Second, we did not include recursive definitions. The rationale for this decision and the alternative construct we use are discussed in Section 3.1.*

## 3 Syntax and Typing Rules

In this section, we will present the extensions we have made to the syntax of the  $\pi$ -calculus, as previously



introduced, in order to represent the processes that are of interest to us. These are processes capable of generating random numbers in order to randomise the output of a program and thereby anonymise the input data to a certain extent. We also introduce typing rules that are very similar to the ones in Honda et al. [24, Figure 1], with the primary distinction being the addition of privacy parameters  $(\epsilon, \delta)$  within the typing judgements.

### 3.1 Process Replication and Random Number Generation

We extend the syntax of processes provided in Figure 2 to include two new constructs: finite process replication (which partially replaces recursive definitions) and random number generation (RNG).

$$P, Q, \dots ::= \dots \mid *_n P \mid \mathbf{Lap}_b k?(x) . P \quad (5)$$

and accordingly the syntax of session types:

$$\alpha ::= \dots \mid *_n \alpha. \quad (6)$$

Note that generating a random number is equivalent, syntactically, to receiving a number from a special channel.

**Remark 3.1** *We allow finite process replication rather than arbitrary replication as we do not want a process to generate an infinite number of random numbers during its execution. Indeed, we aim to formalise without extending the notion of interactive differential privacy found in the literature. For instance, Vadhan and Wang [1, Definition 1.5] generates binary strings before the interaction, and Lyu [2, Section A.1.1] explicitly bounds the number of interaction rounds.*

### 3.2 Typing Judgements

We consider two kinds of typing judgements: Fuzz judgements [11, Section 2.2] for expressions written in a simple functional language

$$\Gamma \vdash_{\text{Fuzz}} e : A$$

where  $\Gamma$  is a linear context mapping variables to types (including function types and recursive types), and process judgements

$$\Gamma \vdash P \triangleright \Delta; (\epsilon, \delta)$$

where  $\Gamma$  is a classical context mapping expression variables and session names to types, written  $A$  and  $\langle \alpha, \bar{\alpha} \rangle$  respectively, (an *environment*),  $\Delta$  is a linear context mapping session channels to session types (a *typing*), and  $(\epsilon, \delta)$  are privacy parameters. We say that a context  $\Delta$  is *completed* when it only contains end types [31, Section 2.2]. Anticipating the semantics, such a judgement reads, “Under the environment  $\Gamma$  and the typing  $\Delta$ ,  $P$  is an  $(\epsilon, \delta)$ -differentially private process.”

We write  $[\Gamma]_s$  for the Fuzz context obtained by adding a sensitivity annotation  $s$  to each expression type in  $\Gamma$  and  $\Gamma_1 \amalg \Gamma_2$  for the disjoint union of  $\Gamma_1$  and  $\Gamma_2$  when they do not have any variable in common.

### 3.3 Operations and Relations

Before we can state the typing rules, we need to define a number of operations and relations on processes and typing contexts.

#### 3.3.1 Congruence

First of all, the sets of free names, free variables and free channels of a process  $P$ , as defined in the standard way (see Honda et al. [24, Section 2.2]), are respectively denoted by  $\text{fn}(P)$ ,  $\text{fv}(P)$  and  $\text{fc}(P)$ . We write  $\text{fu}(P)$  for  $\text{fn}(P) \cup \text{fc}(P)$ . Henceforth, a substitution (of free variables) for  $P$  is a function from the set  $\text{fv}(P)$  to the set of values. A substitution is well-typed for an environment  $\Gamma$  when it maps each free name  $x \in \text{fv}(P)$  to a value that is consistent with the type  $\Gamma(x)$ .

**Definition 3.1** *Strong and weak congruence relations (denoted by  $\equiv$  and  $\cong$ , respectively) are the smallest relations that satisfy the following equations:*

- if  $P \equiv_\alpha Q$  (if  $P$  is  $\alpha$ -equivalent to  $Q$ ), then  $P \equiv Q$ ,

Process	$P, Q, \dots$	$::=$ 0 $ $ $k![e].P$ $ $ $k?(x).P$ $ $ <b>if</b> $e$ <b>then</b> $P$ <b>else</b> $Q$ $ $ $s \triangleright \{l_i : P_i \mid i \in I\}$ $ $ $s \triangleleft l.P$ $ $ $P \parallel Q$ $ $ $(\nu a)P$ $ $ $\bar{a}(k).P$ $ $ $a(k).P$ $ $ $(\nu k)P$	inaction value sending value reception conditional label branching label selection composition name hiding session request session acceptance channel hiding
Session types	$\alpha$	$::=$ $?A.\alpha$ $ $ $!A.\alpha$ $ $ $\&_i\{l_i : \alpha_i\}$ $ $ $\oplus_i\{l_i : \alpha_i\}$ $ $ end	

Figure 2: Syntax of the  $\pi$ -calculus with session types

- $*_0P \equiv 0$ ,
- $(\nu u)0 \equiv 0$ ,  $(\nu uu')P \equiv (\nu u'u)P$ , and if  $u \notin \text{fu } Q \cup \text{fu } Q$ , then  $(\nu u)P \parallel Q \equiv (\nu u)(P \parallel Q)$  and  $Q \parallel (\nu u)P \equiv (\nu u)(Q \parallel P)$ ,
- if  $P \equiv Q$ , then  $P \cong Q$ ,
- $P \parallel 0 \cong P$  and  $0 \parallel P \cong P$ ,
- $P \parallel Q \cong Q \parallel P$  and  $P \parallel (Q \parallel R) \cong (P \parallel Q) \parallel R$ .

**Remark 3.2** *Contrary to Honda et al. [24, Section 2.3], both Honda et al. [30, Figure 2] and Yoshida and Vasconcelos [31, Figure 2] consider that, in general,  $(\nu uu)P$  and  $(\nu u)P$  are not congruent processes. We adhere to the latter works.*

Most  $\pi$ -calculus systems with session types consider only a single congruence relation. However, in our approach, since we do not treat the processes  $P \parallel Q$  and  $Q \parallel P$ , as well as  $P \parallel (Q \parallel R)$  and  $(P \parallel Q) \parallel R$  indifferently (see Definition 4.5 of process view, as well as the typing and reduction rules), we distinguish between strong congruence, for which the operation  $\parallel$  is neither commutative nor associative, and weak congruence. This will lead in Section 4.1 to the introduction of two additional reduction rules, [R-COMM]

and [R-ASSOC]. In like manner,  $P \parallel 0$  and  $P$  are only weakly congruent, as their traces will have different structures: the former is always a proper binary tree, whereas the latter might be reduced to a leaf.

### 3.3.2 Composition of privacy parameters

We define  $\star$ , a binary operation between privacy parameters [1, Theorem 1.8], which will be used in the rule [T-CONC] in the following way:

$$(\epsilon_1, \delta_1) \star (\epsilon_2, \delta_2) = (\epsilon_1 + \epsilon_2, \delta_1 + e^{\epsilon_1} \delta_2). \quad (7)$$

Observe that this operation is neither commutative nor associative.

**Remark 3.3** *In order to obtain tighter bounds, we would consider a family of  $n$ -ary typing rules for concurrent composition, each with a corresponding  $n$ -ary operation on privacy parameters,*

### 3.3.3 Session types

In the communication between two processes over a given channel, they perform symmetrical roles, as each message received by one process is sent by the other, and vice versa. For this reason, it is natural



to define the co-type of a session type. We accomplish this essentially in the same way as described by Honda et al. [24, Definition 5.1], but with the addition of considering the types involved in finite replication.

**Definition 3.2** *The co-type  $\bar{\alpha}$  of a session type  $\alpha$  is defined by structural induction using the following equations: 1.  $\bar{\bar{\alpha}} = \alpha$ , 2.  $\overline{?A}.\alpha = !A.\bar{\alpha}$ , 3.  $\overline{\&_i\{l_i : \alpha_i\}} = \oplus_i\{\bar{l}_i : \bar{\alpha}_i\}$ , 4.  $\overline{*_n\alpha} = *_n\bar{\alpha}$ .*

**Definition 3.3** ([24, Definition 5.2]) *We say that two typings  $\Delta_1$  and  $\Delta_2$ , which map session channels to session types, are compatible whenever for all channels  $k$  in  $\text{dom}(\Delta_1) \cap \text{dom}(\Delta_2)$ , we have  $\Delta_1(k) = \Delta_2(k)$ . In this case, we write  $\Delta_1 \asymp \Delta_2$ .*

**Definition 3.4** ([24, Definition 5.2]) *The composition  $\Delta_1 \circ \Delta_2$  of two compatible typings  $\Delta_1$  and  $\Delta_2$  is defined by*

$$\begin{aligned} & (\Delta_1 \circ \Delta_2)(k) \\ &= \begin{cases} \perp & \text{if } k \in \text{dom}(\Delta_1) \cap \text{dom}(\Delta_2) \\ \Delta_i(k) & \text{if } k \in \text{dom}(\Delta_i) \setminus \text{dom}(\Delta_{3-i}) \end{cases} \quad (8) \end{aligned}$$

### 3.4 Typing Rules

The typing rules for our process calculus are now given in Figure 3. Most of them are standard for a process calculus with session types (see Honda et al. [24, Figure 1] and Yoshida and Vasconcelos [31, Figure 6]), except for the addition of privacy annotations, which form a second linear context, alongside the session context.

However, we draw the reader's attention to the following rules: [T-NSEND] for the Laplace mechanism; [T-CONC] for concurrent composition, [T-PAR] for parallel composition, and [T-REP] for the finite replication. Moreover, what was stated as a theorem in Honda et al. [24, Theorem 5.4 (1)], assuming the inclusion of the [T-BOT] typing rule from Yoshida and Vasconcelos [31, Section 2.3], due to the non-associativity of the  $\star$  law used in [T-CONC] has to be enforced in our case by the [T-CONG] structural typing rule.

**Example 3.1** *As a running example, let us consider an implementation of the one-round Laplace mechanism for a fixed privacy parameter  $\epsilon$ . Specifically, we define*

$$\begin{aligned} M &= k?(f) . \mathbf{Lap}_{1/\epsilon}?(r) . k![f(D) + r] . \text{end} \\ A &= k![f] . k?(y) . \text{end} . \end{aligned}$$

*The following typing judgements are derivable for  $\alpha = ?(\text{db} \multimap \text{Num}) . !\text{Num} . \text{end}$ ,*

$$\begin{aligned} D : \text{Db}, f : \text{Db} \multimap \text{Num} \vdash M \triangleright k : \alpha; (\epsilon, 0) \\ f : \text{Db} \multimap \text{Num} \vdash A \triangleright k : \bar{\alpha}; (0, 1) \end{aligned}$$

*In order to obtain closed programs, we would need to abstract the channels using [T-ACC] and [T-REQ], and add therefore a session name in the environment. However, since we are focusing on communication here, this is not necessary.*

In the typing rules as well as in the previous example, either because the process we are considering is not meant to be combined with an adversary (as it is an adversary itself) or because the process does not preserve privacy, we chose  $(0, 1)$  as a neutral value for the privacy parameters. This choice is always sound in the sense of Section 4.3, as it is sound if and only if a certain probability is less than 1.

An example of a typing rule that does not preserve privacy is [T-IF] for conditional expressions. Intuitively, even if  $D$  and  $D'$  are adjacent databases, the processes **if**  $x \in D$  **then**  $P$  **else**  $Q$  and **if**  $x \notin D$  **then**  $P$  **else**  $Q$  can have arbitrarily different behaviour.

Similarly, sending data as in the rule [T-SEND] does not ensure differential privacy because the data could be private and might even include the entire dataset. Therefore, to achieve meaningful privacy parameters, it is necessary to use a rule like [T-NSEND], which requires the process to first add noise to the data based on its sensitivity relative to the dataset. The sensitivity analysis for this process is conducted using the Fuzz calculus.

Finally, our type system has a weakening property that allows one to effectively use the [T-PAR] rule

$$\begin{array}{c}
\frac{\Delta \text{ completed}}{\Gamma \vdash 0 \triangleright \Delta; (\epsilon, \delta)} \text{ [T-INACT]} \quad \frac{\Gamma \vdash P \triangleright \Delta, k : \text{end}; (\epsilon, \delta)}{\Gamma \vdash P \triangleright \Delta, k : \perp; (\epsilon, \delta)} \text{ [T-BOT]} \\
\frac{\Gamma \vdash a : \langle \alpha, \bar{\alpha} \rangle \quad \Gamma \vdash P \triangleright \Delta, k : \alpha; (\epsilon, \delta)}{\Gamma \vdash a(k) . P \triangleright \Delta; (\epsilon, \delta)} \text{ [T-ACC]} \quad \frac{\Gamma \vdash a : \langle \alpha, \bar{\alpha} \rangle \quad \Gamma \vdash P \triangleright \Delta, k : \bar{\alpha}; (\epsilon, \delta)}{\Gamma \vdash \bar{a}(k) . P \triangleright \Delta; (\epsilon, \delta)} \text{ [T-REQ]} \\
\frac{[\Gamma]_{+\infty} \vdash_{\text{Fuzz}} e : A \quad \Gamma \vdash P \triangleright \Delta, k : \alpha; (\epsilon, \delta)}{\Gamma \vdash k![e] . P \triangleright \Delta, k : !A . \alpha; (0, 1)} \text{ [T-SEND]} \\
\frac{[\Gamma]_s \vdash_{\text{Fuzz}} e : \text{Num} \quad \Gamma \vdash P \triangleright \Delta, k : \alpha; (\epsilon, \delta)}{\Gamma \vdash \mathbf{Lap}_{s/\epsilon'}?(x) . k![e+x] . P \triangleright \Delta, k : !\text{Num} . \alpha; (\epsilon + \epsilon', \delta)} \text{ [T-NSEND]} \\
\frac{\Gamma, x : A \vdash P \triangleright \Delta, k : \alpha; (\epsilon, \delta)}{\Gamma \vdash k?(x) . P \triangleright \Delta, k : ?A . \alpha; (\epsilon, \delta)} \text{ [T-RCV]} \quad \frac{\Gamma, x : \text{Num} \vdash P \triangleright \Delta; (\epsilon, \delta) \quad \vdash_{\text{Fuzz}} b : \text{Num}}{\Gamma \vdash \mathbf{Lap}_b?(x) . P \triangleright \Delta; (\epsilon, \delta)} \text{ [T-LAP]} \\
\frac{[\Gamma]_s \vdash_{\text{Fuzz}} e : \text{Bool} \quad \Gamma \vdash P \triangleright \Delta; (\epsilon_P, \delta_P) \quad \Gamma \vdash Q \triangleright \Delta; (\epsilon_Q, \delta_Q)}{\Gamma \vdash \mathbf{if } e \mathbf{ then } P \mathbf{ else } Q \triangleright \Delta; (0, 1)} \text{ [T-IF]} \\
\frac{\Gamma \vdash P_1 \triangleright \Delta, k : \alpha_1; (\epsilon, \delta) \quad \dots \quad \Gamma \vdash P_n \triangleright \Delta, k : \alpha_n; (\epsilon, \delta)}{\Gamma \vdash k \triangleright \{l_i : P_i\}_{i \in I} \triangleright \Delta, k : \&_{i \in I} \{l_i : \alpha_i\}; (\epsilon, \delta)} \text{ [T-BR]} \\
\frac{\Gamma \vdash P \triangleright \Delta, k : \alpha_j; (\epsilon, \delta) \quad j \in I}{\Gamma \vdash k \triangleleft l_j . P \triangleright \Delta, k : \oplus_{i \in I} \{l_i : \alpha_i\}; (\epsilon, \delta)} \text{ [T-SEL}_j\text{]} \\
\frac{\Gamma \vdash P_1 \triangleright \Delta_1; (\epsilon_1, \delta_1) \quad \Gamma \vdash P_2 \triangleright \Delta_2; (\epsilon_2, \delta_2) \quad \Delta_1 \asymp \Delta_2}{\Gamma \vdash P_1 \parallel P_2 \triangleright \Delta_1 \circ \Delta_2; (\epsilon_1, \delta_1) \star (\epsilon_2, \delta_2)} \text{ [T-CONC]} \\
\frac{\Gamma_1 \vdash P_1 \triangleright \Delta_1; (\epsilon, 0) \quad \Gamma_2 \vdash P_2 \triangleright \Delta_2; (\epsilon, 0) \quad \Delta_1 \asymp \Delta_2}{\Gamma_1 \amalg \Gamma_2 \vdash P_1 \parallel P_2 \triangleright \Delta_1 \circ \Delta_2; (\epsilon, 0)} \text{ [T-PAR]} \\
\frac{\Gamma, a : A \vdash P \triangleright \Delta; (\epsilon, \delta)}{\Gamma \vdash (\nu a)P \triangleright \Delta; (\epsilon, \delta)} \text{ [T-NHIDE]} \quad \frac{\Gamma \vdash P \triangleright \Delta, k : \perp; (\epsilon, \delta)}{\Gamma \vdash (\nu k)P \triangleright \Delta; (\epsilon, \delta)} \text{ [T-CHIDE]} \\
\frac{\Gamma \vdash P \triangleright \Delta; (\epsilon, \delta) \quad P \equiv Q}{\Gamma \vdash Q \triangleright \Delta; (\epsilon, \delta)} \text{ [T-CONG]} \quad \frac{\Gamma \vdash P \triangleright \Delta \quad \vdash_{\text{Fuzz}} n : \mathbb{N}at}{\Gamma \vdash *_n P \triangleright *_n \Delta; (n\epsilon, \delta \frac{e^{n\epsilon} - 1}{\epsilon})} \text{ [T-REP]}
\end{array}$$

Figure 3: Process Typing Rules

with different privacy parameters in its premises or the [T-CONC] rule with some variables used in only one of the composed processes.

**Theorem 3.1 (Weakening)** *If  $x \notin \Gamma$ ,  $a \notin \Gamma$ , and  $\Gamma \vdash P \triangleright \Delta$ ;  $(\epsilon, \delta)$ , then  $\Gamma, x : A, a : \langle \alpha, \bar{\alpha} \rangle \vdash P \triangleright \Delta$ ;  $(\epsilon + \epsilon', \delta + \delta')$ .*

## 4 Operational Semantics

### 4.1 Reduction Rules

We will provide an operational semantics for our process calculus, presented as a fully probabilistic labelled transition system (FPLTS). We consider a *fully* probabilistic system as the session types ensure the absence of non-probabilistic non-determinism during the reduction of a process. Moreover, the labels will trace the execution, which we will use to define the view of a party during an interaction.

**Definition 4.1 ([32, Definition 6.1])** *A fully probabilistic labelled transition system (FPLTS) is a triple  $(S, A, P)$  where*

- $S$  is a set of states (processes),
- $A$  is a countable set of labels (actions),
- $P : S \times A \times S \rightarrow [0, 1]$  is a transition probability function such that for all states  $x$ ,

$$\sum_{\alpha \in A} \sum_{t \in T} P(s, \alpha, t) \in \{0, 1\}. \quad (9)$$

To write the rules for such a transition system, we use the notation introduced by Chatzikokolakis and Palamidessi [33, Figure 1] for their process calculus  $\pi_{\text{prob}}$ . Specifically, we write

$$P \left\{ \frac{l_i}{p_i} \rightarrow Q_i \right\}_{i \in I}$$

when for all  $i$  in  $I$ , a process  $P$  reduces to  $Q_i$  with probability  $p_i$  and label  $l_i$ .

In our case, the labels are binary trees labelled at the leaves by atoms. These atoms are intended to trace one reduction step of a simple process, that

is one that does not involve the parallel construct. An atom can be, for example,  $\epsilon_v$  for a conditional evaluating to a boolean value  $v$ , or  $\alpha_{n,v}$  for a value exchange during an  $n$ -repetition (we may omit the first element of the tuple when it is 1). Atoms can be concatenated using the  $+$  operator<sup>1</sup> to form *lists* of atoms, a fact which will be helpful later when defining multi-step reduction.

The complete set of reduction rules is presented in Figure 4.

Note that [R-LAP], as expected, is the only rule that introduces randomness, and it does so through the discrete Laplace distribution. The other rules merely propagate this randomness.

**Definition 4.2 (Ghosh et al. [25, Example 2.1])** *The discrete Laplace distribution with parameter  $b > 0$ , is defined by the following probability mass function over  $\mathbf{Z}$ :*

$$n \mapsto \underbrace{\frac{e^{1/b} - 1}{e^{1/b} + 1}}_{p_{n,b}} \times e^{-|n|/b}. \quad (10)$$

### 4.2 Trace and View of a Process

Now, we will define the *view of a process* during an interaction, which is an essential component, as we have seen, in the definition of interactive differential privacy. To achieve this, we will begin by defining the multi-step reduction relation and the trace of an execution.

**Lemma 4.1** *If  $P \left\{ \frac{l_i}{p_i} \rightarrow Q_i \right\}_i$ , and for all  $i$  we have  $Q_i \left\{ \frac{l'_j}{p'_j} \rightarrow R_j \right\}_j$ , then for all  $i$  and  $j$ , the trees  $k_i$  and  $l_j$  have the same shape.*

*Proof.* The only notable case is that of the [R-CONG] reduction rule. We have explicitly ensured in Definition 3.1 that strong congruence preserves the shape of processes.  $\square$

This lemma allows us to define the two-step reduction relation using tree concatenation as follows.

<sup>1</sup>Consider the `cons` operator from languages in the LISP family.

$$\begin{array}{c}
\frac{e \downarrow \top}{\text{if } e \text{ then } P \text{ else } Q \left\{ \frac{\epsilon_{\top}}{1} \rightarrow P \right\}} \quad [\text{R-TRUE}] \qquad \frac{e \downarrow \perp}{\text{if } e \text{ then } P \text{ else } Q \left\{ \frac{\epsilon_{\perp}}{1} \rightarrow P \right\}} \quad [\text{R-FALSE}] \\
\\
\frac{e \downarrow v}{k![e] . P \parallel k?(x) . Q \left\{ \frac{(\alpha_1, v, \alpha_1, v)}{1} \rightarrow P \parallel Q[v/x] \right\}} \quad [\text{R-VAL}] \qquad \frac{}{a(k) . P \parallel \bar{a}(k) . Q \left\{ \frac{(\chi_k, \chi_k)}{1} \rightarrow (\nu k)(P \parallel Q) \right\}} \quad [\text{R-CHAN}] \\
\\
\frac{}{k \triangleleft l_i . P \parallel k \triangleright \{l_i : P_i\}_i \left\{ \frac{(\delta_i, \delta_i)}{1} \rightarrow P \parallel P_i \right\}} \quad [\text{R-SEL}] \\
\\
\frac{P \left\{ \frac{l_i}{p_i} \rightarrow P_i \right\}_i}{(\nu u) P \left\{ \frac{l_i}{p_i} \rightarrow (\nu u) P_i \right\}_i} \quad [\text{R-HIDE}] \qquad \frac{P \left\{ \frac{l_i}{p_i} \rightarrow P_i \right\}_i}{P \parallel Q \left\{ \frac{(l_i, \emptyset)}{p_i} \rightarrow P_i \parallel Q \right\}_i} \quad [\text{R-CONC}] \\
\\
\frac{}{\mathbf{Lap}_b?(x) . P \left\{ \frac{\gamma_n}{p_n, b} \rightarrow P[n/x] \right\}_{n \in \mathbf{Z}}} \quad [\text{R-LAP}] \\
\\
\frac{e \downarrow v}{k![e] . P_1 \parallel *_{n} k?(x) . P_2 \left\{ \frac{(\alpha_n, v, \alpha_n, v)}{1} \rightarrow P_1 \parallel (P_2[v/x] \parallel *_{n-1} k?(x) . P_2) \right\}} \quad [\text{R-REP}] \\
\\
\frac{P' \left\{ \frac{l_i}{p_i} \rightarrow Q'_i \right\}_i \quad P \equiv P' \quad (\forall i)(Q_i \equiv Q'_i)}{P \left\{ \frac{l_i}{p_i} \rightarrow Q_i \right\}_i} \quad [\text{R-CONG}] \\
\\
\frac{P_1 \parallel P_2 \left\{ \frac{(l_i, r_i)}{p_i} \rightarrow Q_{i_1} \parallel Q_{i_2} \right\}_i}{P_2 \parallel P_1 \left\{ \frac{(r_i, l_i)}{p_i} \rightarrow Q_{i_2} \parallel Q_{i_1} \right\}_i} \quad [\text{R-COMM}] \qquad \frac{P_1 \parallel (P_2 \parallel P_3) \left\{ \frac{(l_1, (l_2, l_3))}{p_i} \rightarrow P_{1i} \parallel (P_{2i} \parallel P_{3i}) \right\}_i}{(P_1 \parallel P_2) \parallel P_3 \left\{ \frac{((l_1, l_2), l_3)}{p_i} \rightarrow (P_{1i} \parallel P_{2i}) \parallel P_{3i} \right\}_i} \quad [\text{R-ASSOC}]
\end{array}$$

Figure 4: Process Reduction Rules

**Definition 4.3** The concatenation  $t_1 + t_2$  of two binary trees  $t_1$  and  $t_2$ , which are labelled at the leaves and have the same shape, is the binary tree obtained by concatenating the labels leaf by leaf.

**Example 4.1** The concatenation of  $(a, (b, c))$  and  $(d, (e, [f, g]))$  is the tree  $([a, d], ([b, e], [c, f, g]))$ .

**Definition 4.4** For any two processes  $P$  and  $Q$ , let  $X_{PR}$  be the set of all processes  $R$  such that  $P$  can reduce to  $R$  with probability  $p_{PQ}$  and label  $l_{PQ}$ , and  $Q$  can reduce to  $R$  with probability  $p_{QR}$  and label  $l_{QR}$ . Now, for a given label  $l$ , let  $X_{PR,l}$  be the subset of  $X_{PR}$  that consists of the processes  $R$  such that  $l_{PQ} + l_{QR} = l$ . We say that  $P$  reduces in two steps to  $R$  with probability  $p$  and label  $l$  whenever  $X_{PR,l}$  is non-empty and  $p = \sum_{Q \in X_{PR,l}} p_{PQ} \cdot p_{QR}$ . We then write

$$P \left\{ \frac{l_i}{p_i} \twoheadrightarrow^2 R_i \right\}$$

to account for all such  $R$ .

This construction generalises by induction to any number of steps, and we write  $\longrightarrow^*$  for the transitive and reflexive closure of  $\longrightarrow$ . We then call a possible label for this multi-step reduction a *trace*—which is a binary tree labelled at the leaves with lists of atoms—and we denote by  $\mathbf{T}$  the set of all possible traces.

Finally, given an appropriate countable probability space  $\Omega$ , the trace  $\text{Tr}(P)$  of the execution of a process  $P$ , where

$$P \left\{ \frac{l_i}{p_i} \twoheadrightarrow^* P_i \right\}_i,$$

is the naturally defined random variable such that for all  $i$

$$\Pr[\text{Tr}(P) = l_i] = p_i.$$

**Remark 4.1** The definition of traces we have provided is quite intricate and subtle, as it involves using trees instead of lists. For an explanation involving lists, see, for example, Bian and Abate [34, Definition 6].

In order to gain some intuition on the behaviour of traces, consider the following lemma.

**Lemma 4.2** For all processes  $P$  and  $Q$ , the probability that  $\text{Tr}(P \parallel Q)$  is a leaf equals 0.

*Proof.* The only reduction rules that can apply to a concurrent composition are [R-VAL], [R-CHAN], [R-SEL], [R-REP], and the structural rules [R-COMM] and [R-ASSOC]. The labels of the conclusion of each of these rules form a binary tree with two subtrees.  $\square$

Now, we define the view of a party during an interaction as the left subtree of the trace of the execution.

**Definition 4.5** The view of a process  $A$  interacting with a process  $M$ , written as  $\text{View}(A \parallel M)$ , is the following random variable:  $\text{Left}(\text{Tr}(A \parallel M))$ , that is,  $\omega \mapsto \text{Left}(\text{Tr}(A \parallel M)(\omega))$ .

**Remark 4.2** Except in rare particular cases, we have  $\text{View}(A \parallel M) \neq \text{View}(M \parallel A)$ .

**Example 4.2** Taking the same example as before (Example 3.1), for every integer  $n$ ,

$$\Pr[\text{Tr}(A[f] \parallel M[D]) = (l_n, r_n)] = p_{n,1/\epsilon},$$

where  $l_n = [\alpha_f, \alpha_{f(D)+n}]$ , and  $r_n = [\alpha_f, \gamma_n, \alpha_{f(D)+n}]$ . In addition, for all  $n$ , we have  $\Pr[\text{View}(A \parallel M) = l_n] = p_{n,1/\epsilon}$  and  $\Pr[\text{View}(M \parallel A) = r_n] = p_{n,1/\epsilon}$ . Note that by summing over all integers  $n$ , we find that the probability that the trace of this execution and view of either party is of the given form equals 1. In particular, in contrast to  $M$ , which always has access to it, as shown by the presence of  $\gamma_n$  in its view, the adversary  $A$  never has access to the generated random value. This prevents them from subtracting the value from the final result to infer private information.

### 4.3 Differential Privacy as Approximate Trace Equivalence

We say that a process  $M$  is differentially private when  $M[S]$  and  $M[S']$  have approximately the same trace when interacting with an adversary, provided that  $S$  and  $S'$  are sufficiently close.

**Definition 4.6** Given an environment  $\Gamma$  and a typing  $\Delta$ , we say that a process  $M$  is  $(\epsilon, \delta)$ -differentially

private if for all adjacent substitutions  $S$  and  $S'$  that are well-typed for  $\Gamma$ , for all adversary processes  $A$  such that  $A \parallel M[S]$  (and  $A \parallel M[S']$ ) are closed processes, and all  $X \subseteq \text{Range}(\mathcal{M})$ , we have the following inequality:

$$\begin{aligned} & \Pr[\text{View}(A \parallel M[S]) \in X] \\ & \leq e^\epsilon \Pr[\text{View}(A \parallel M[S']) \in X] + \delta. \end{aligned} \quad (11)$$

In practice, for straightforward cases,  $S$  and  $S'$  will often be substitutions involving a single variable. The entire database and the adjacency of two substitutions thus correspond to the adjacency of their associated databases. However, this relation is subject to redefinition depending on the specific area of application.

**Remark 4.3** *Baier et al. [32, Theorem 4] have shown that their notion of approximate trace equivalence for labelled Markov chains is induced by a form of approximate probabilistic bisimulation. We therefore expect that one could rephrase our definition of interactive differential privacy in the latter framework and obtain the same metatheoretical properties as those in Section 5.*

## 5 Metatheoretical Properties

We must prove that our type system provides sufficient conditions to ensure interactive differential privacy for processes (according to Definition 4.6); in other words we need to show its soundness. To achieve this, the most crucial part is to show that the choice of privacy parameters is sound.

Let us begin by stating some properties regarding the traces and views of processes.

**Lemma 5.1** *For all processes  $P$ ,  $Q$  and  $R$  and traces  $t_1$ ,  $t_2$  and  $t_3$ , we have*

$$\begin{aligned} & \Pr[\text{Tr}((P \parallel Q) \parallel R) = ((t_1, t_2), t_3)] \\ & = \Pr[\text{Tr}(P \parallel (Q \parallel R)) = (t_1, (t_2, t_3))]. \end{aligned} \quad (12)$$

*Proof.* This is a consequence of the structural reduction rule [R-ASSOC].  $\square$

We write  $\text{Left}^*(T)$  for the inverse image of  $T$  under the function  $\text{Left}$ , which is the set  $\{t \in \mathbf{T} \mid \text{Left}(t) \in T\}$ .

**Lemma 5.2** *For all processes  $P$ ,  $Q$  and  $R$  and sets of traces  $T$ , we have*

$$\begin{aligned} & \Pr[\text{View}((P \parallel Q) \parallel R) \in \text{Left}^*(T)] \\ & = \Pr[\text{View}(P \parallel (Q \parallel R)) \in T]. \end{aligned} \quad (13)$$

*Proof.*

$$\begin{aligned} & \Pr[\text{View}((P \parallel Q) \parallel R) \in \text{Left}^*(T)] \\ & = \sum_{t_1 \in \text{Left}^*(T)} \sum_{t_2 \in \mathbf{T}} \Pr[\text{Tr}((P \parallel Q) \parallel R) = (t_1, t_2)] \\ & = \sum_{t_1 \in T} \sum_{t_2 \in \mathbf{T}} \sum_{t_3 \in \mathbf{T}} \Pr[\text{Tr}((P \parallel Q) \parallel R) = ((t_1, t_2), t_3)] \\ & = \sum_{t_1 \in T} \sum_{t_2 \in \mathbf{T}} \sum_{t_3 \in \mathbf{T}} \Pr[\text{Tr}(P \parallel (Q \parallel R)) = (t_1, (t_2, t_3))] \\ & = \sum_{t_1 \in T} \sum_{t_2 \in \mathbf{T}} \Pr[\text{Tr}(P \parallel (Q \parallel R)) = (t_1, t_2)] \\ & = \Pr[\text{View}(P \parallel (Q \parallel R)) \in T]. \end{aligned}$$

Note that we use Lemma 5.1 to prove the third equality.  $\square$

**Corollary 5.3** *The [T-CONC] typing rule is sound.*

The proof by [Vadhan and Wang](#) of the concurrent composition theorem becomes more straightforward in our framework, given that the ad hoc post-processing step on the view of the combined adversary is reduced to merely taking its left subtree.

*Proof.* Without loss of generality, we restrict our analysis to the case where  $\delta_1 = \delta_2 = 0$ , see [1, Theorem 3.3]. Let  $S$  and  $S'$  be two adjacent substitutions, we want to show that for all sets  $T$ ,

$$\begin{aligned} & \Pr[\text{View}(A \parallel (M_1[S] \parallel M_2[S])) \in T] \\ & \leq e^{\epsilon_1 + \epsilon_2} \Pr[\text{View}(A \parallel (M_1[S'] \parallel M_2[S'])) \in T]. \end{aligned}$$

To this end, since  $e^{\epsilon_1} e^{\epsilon_2} = e^{\epsilon_1 + \epsilon_2}$ , it suffices to show that we can perform one substitution at a time, that is,

$$\begin{aligned} & \Pr[\text{View}(A \parallel (M_1[S] \parallel M_2[S])) \in T] \\ & \leq e^{\epsilon_1} \Pr[\text{View}(A \parallel (M_1[S'] \parallel M_2[S])) \in T]. \end{aligned}$$



This inequality indeed holds, which we will prove by introducing a combined adversary and using Lemma 5.2:

$$\begin{aligned}
& \Pr[\text{View}(A \parallel (M_1[S] \parallel M_2[S])) \in T] \\
&= \Pr[\text{View}((A \parallel M_1[S]) \parallel M_2[S]) \in \text{Left}^*(T)] \\
&\leq e^{\epsilon^2} \Pr[\text{View}((A \parallel M_1[S]) \parallel M_2[S']) \in \text{Left}^*(T)] \\
&= e^{\epsilon^2} \Pr[\text{View}(A \parallel (M_1[S] \parallel M_2[S'])) \in T]
\end{aligned}$$

The inequality arises from our hypothesis that  $M_2$  is an  $\epsilon_2$ -differentially private process.  $\square$

**Lemma 5.4** *The [T-PAR] typing rule is sound.*

*Proof.* See Theorem 2.2.  $\square$

**Lemma 5.5** *The [T-REP] typing rule is sound.*

*Proof.* This result follows from Vadhan and Wang [1, Theorem 1.8], who obtain better privacy parameters than those achieved by repeatedly applying the  $\star$  law, as discussed in Section 3.3.2.  $\square$

Finally, we can the soundness theorem and the same good properties for our language as Honda et al. [24, Theorem 5.4].

**Theorem 5.6 (Soundness)** *If  $\Gamma \vdash M \triangleright \Delta; (\epsilon, \delta)$ , then  $M$  is an  $(\epsilon, \delta)$ -differentially private process.*

This implies that if a process  $M$  is well-typed, then it is  $(\epsilon, \delta)$ -differentially private (as defined in Definition 4.6), and discloses only a limited amount of private information when interacting with *any* adversary  $A$  that meets the given assumptions. This is particularly notable as the well-typedness of the process is proved by a finite derivation.

**Theorem 5.7**

- *Typing is preserved by reduction:*  
If  $\Gamma \vdash P \triangleright \Delta; (\epsilon, \delta)$  and  $P \left\{ \frac{l_i}{p_i} \star Q_i \right\}$ , then for all  $i$ , we have  $\Gamma \vdash Q_i \triangleright \Delta; (\epsilon, \delta)$ .
- *A typable program never reduces into an error.*

To be precise, the second point means that the type system ensures communication safety (only data of the expected type are exchanged) and session fidelity [29, Section 2].

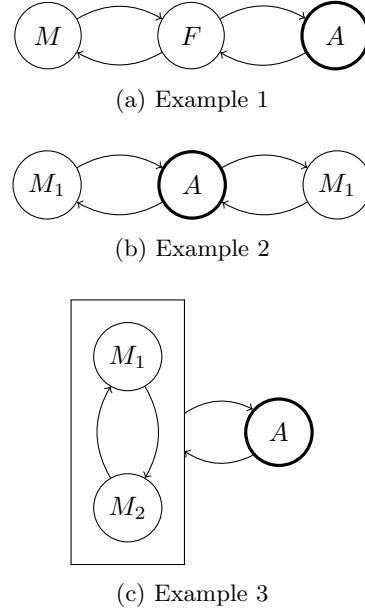


Figure 5: Interaction between the different processes of the example section

## 6 Examples

### 6.1 The Forwarder Process

To demonstrate how our type system accounts for communication between mechanisms themselves, and not merely the interaction between mechanisms and an adversary, let us consider the scenario where a process that does not own any private data acts as an intermediary between a mechanism and an adversary (see Figure 5a) that is a man in the middle (MITM) for our running example (Example 3.1).

Given a 1-Lipschitz function (also known as a 1-sensitive function) from  $\text{Db}$  to  $\text{Num}$ , a database  $D$ , and channels  $k$  and  $k'$  of session type  $\alpha = ?(\text{Db} \multimap \text{Num}) . !\text{Num} . \text{end}$ , or of its co-type  $\bar{\alpha}$ , we define

$$\begin{aligned}
A &= k'![f] . k'?(x) . 0 \\
F &= k'?(f) . k![f] . k?(x) . k'![x] . 0 \\
M &= k?(f) . \mathbf{Lap}_{1/\epsilon}?(r) . k![f(D) + r] . 0 .
\end{aligned}$$

Even though, when reducing  $A[f] \parallel (F \parallel M[D])$ , the adversary  $A$ , and indeed every possible well-typed ad-

versary, only interacts with  $F$ , which owns no private information and is therefore differentially private for all privacy parameters,  $F \parallel M[D]$  is correctly typed as  $\epsilon$ -differentially private by our typing rules.

## 6.2 A Database Split between Two Servers

To illustrate the rule of parallel composition and the fact that a substitution for a mechanism may involve more than one variable, let us consider two processes  $M_1$  and  $M_2$  that exhibit identical behaviour (for instance, both implementing the Laplace mechanism) but have access to different halves of the database, and a data analyst who wants to roughly estimate the total number of entries. By applying [T-PAR], we get

$$D_1 : \text{Db}, D_2 : \text{Db} \vdash M_1 \parallel M_2 \triangleright k_1 : \alpha, k_2 : \alpha, k : \beta; (\epsilon, 0)$$

for the straightforward session types  $\alpha$  and  $\beta$ .

The validity of this judgement involves well-typed adjacent substitutions. In this context, a substitution for  $M_1 \parallel M_2$  consists of two databases,  $D_1$  and  $D_2$ . To achieve the correct notion of adjacency for the combined databases, these halves must be disjoint sets.

Let  $\text{count}$  be the 1-sensitive function that counts the number of elements in a set. The analyst may be represented by the following process, using a special channel  $k$  as an output to a device like a screen:

$$A = k_1![\text{count}].k_2![\text{count}].k_1?(c_1).k_2?(c_2).k![c_1+c_2].0$$

(All five other possible orderings between message sending and receiving are well-typed, which illustrates that an adversary can arbitrarily interleave its queries to the two mechanisms [1, Section 1.4].)

## 6.3 The Private Guess-and-Check Algorithm

Let us show that our language is expressive enough to write the Guess-and-Check algorithm (see Algorithm 1, taken from Lyu [2, Algorithm 1]), and that our type system can prove it preserves privacy (see Theorem 6.1).

Let us begin by informally describing the algorithm. An adversary sends queries—which consist of a function to be executed on the database and an expected response—to a central server. If the guess is approximately correct, then the adversary is informed and no privacy budget is consumed. Conversely, if the guess is far from the correct answer, the adversary is informed and provided with a value that is approximately correct. The interaction ceases after a certain number of rounds, unless the privacy budget is completely depleted beforehand.

**Data:** private dataset  $X$ , error tolerance  $E \geq 0$ , privacy parameter  $\epsilon > 0$ , maximum number of negative queries  $c \geq 1$ , number of interaction rounds  $T$

```

 $\rho \leftarrow \text{Lap}(1/\epsilon);$ 
for  $i = 1, 2, \dots, T$  do
  Receive the next query  $(f_i, \tau_i);$ 
   $\gamma_i \leftarrow \text{Lap}(c/\epsilon);$ 
  if  $|f_i(X) - \tau_i| + \gamma_i \geq E + \rho$  then
     $v_i \leftarrow f(X) + \text{Lap}(c/\epsilon);$ 
    report (wrong,  $v_i$ );
     $t \leftarrow t + 1;$ 
    if  $t = c$  then halt;
  else
    | report pass
  end
end

```

**Algorithm 1:** Private Guess-and-Check [2, Algorithm 1]

**Remark 6.1** *In the following, it will be more convenient to use a functional syntax similar to the ML programming language, rather than mathematical syntax.*

This algorithm can be viewed as a variant of the Sparse Vector Technique [18, 19]. As a result, the latter can serve as a foundational component to simulate the Guess-and-Check algorithm.

The Sparse Vector Technique responds to at most  $c$  incorrect queries (for an error tolerance of  $E$ ) out of  $N$  total queries about a database  $D$ . It interacts

on a channel  $k$  and uses a memory cell named  $a$  to store its internal state. A memory cell is an object with two methods named `read` and `write`. For a detailed definition in terms of processes that use the branching and selection constructs, refer to Honda et al. [24, Example 3.3, Example 3.4].

```

SVT(c, E, N, D, k, a) =
  Lap(1/epsilon)?(rho);
  a.write 1;
  repeat N times
    k?(f, v);
    let t = a.read () in
      if t >= c then k![0] else
        Lap(c/epsilon)?(gamma);
        k![abs (f(D) - v) + gamma < E + rho]
      end
    a.write (t + 1);
  end
end

```

Given the environment  $\Gamma = \{c : \text{Nat}, E : \text{Num}, N : \text{Nat}, D : \text{Db}, a : \text{Cell}(\text{Nat})\}$  and the typing  $\Delta = \{k : *_N(?(\text{Db} \multimap \text{Num} \otimes \text{Num}).!\text{Num}.\text{end})\}$ , the following typing rule is sound [19, Theorem 2]:

$$\frac{}{\Gamma \vdash \text{SVT}(c, E, N, D, k, a) \triangleright \Delta; (3\epsilon, 0)} \text{[T-SVT]} \quad (14)$$

Note the crucial usage of a type of the form  $A \multimap B$  for 1-Lipschitz functions from  $\llbracket A \rrbracket$  to  $\llbracket B \rrbracket$ , which is present in Fuzz [11, Section 2.1] but absent in a standard simply typed  $\lambda$ -calculus.

From there, we can follow the reasoning of Lyu [2, Appendix B].

**Theorem 6.1** ([2, Theorem 5]) *The Guess-and-Check algorithm is  $4\epsilon$ -differentially private.*

*Proof.* The Guess-and-Check algorithm can be simulated with  $\text{SVT} \parallel L$  (with the appropriate substitution for the free variables) where  $L$  implements the Laplace mechanisms. Moreover,  $(3\epsilon, 0) \star (\epsilon, 0) = (4\epsilon, 0)$ .  $\square$

## 7 Related work

We have already mentioned in the introduction some references to type systems for non-interactive differential privacy of functional programs, including the

Fuzz language [11] and its extensions. A generalisation of this approach to handle interactive differential privacy has been proposed in Winograd-Cort et al. [35], it consists in a two-layer language and does not deal with any concurrency aspects.

Concerning probabilistic versions of the  $\pi$ -calculus, Chatzikokolakis and Palamidessi [33] defined the  $\pi_{\text{prob}}$ -calculus, a variant of  $\pi$ -calculus with probabilistic choice, and Das et al. [36] introduced a language for probabilistic session types, and similarly to  $\pi_{\text{prob}}$  uses a construction for probabilistic branching. However none of these two papers is motivated by differential privacy. Our approach differs in that we substitute the probabilistic choice with a mechanism for generating random numbers. This is crucial for our type system, as it allows us to manage non-trivial  $(\epsilon, \delta)$  bounds, which would not be possible with probabilistic choice.

Xu [37] investigated DP properties in a probabilistic process calculus, but this calculus is a variant of CCS, a less expressive language. In particular it does not explicitly represent mechanisms such as the Laplace mechanisms.

Some other works like [38, 39] explore DP properties for probabilistic labelled transition systems and relate them to some bisimilarity distances [39]. However their focus is quite different as they do not consider a process syntax as we do in the present paper, and they do not address concurrent interactive DP.

## 8 Future Work

First, we would like to determine whether our calculus, or an extension of it, could guarantee privacy properties in other contexts, particularly in the realm of local differential privacy [40, 41] where there is no central aggregator.

Moreover, if future research in the field of interactive differential privacy were to explore the generation of an infinite sequence of random numbers instead of just a finite one (see Section 3.1) proposing a new definition of a private process, and if they proved composition theorems for this new definition, we could replace our replication primitive with recursive definitions. This would raise interesting seman-

tic issues, particularly in relation to measure theory, since the probability spaces involved would have the cardinality of the continuum.

Finally, as outlined in the section on the semantics of our system, one may investigate the consequences of replacing our current formal definition of interactive differential privacy, which is expressed in terms of approximate trace equivalence, with an alternative framework based on approximate bisimulation.

## References

- [1] S. Vadhan and T. Wang, “Concurrent composition of differential privacy,” in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, vol. 13043. Springer, 2021, pp. 582–604.
- [2] X. Lyu, “Composition theorems for interactive differential privacy,” in *NIPS’22: Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022, pp. 9700–9712.
- [3] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating noise to sensitivity in private data analysis,” in *Theory of Cryptography*, ser. Lecture Notes in Computer Science, vol. 3876. Springer, 2006, pp. 265–284.
- [4] C. Dwork and A. Roth, “The algorithmic foundations of differential privacy,” *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.
- [5] I. Mironov, “Rényi differential privacy,” in *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*. IEEE Computer Society, 2017, pp. 263–275. [Online]. Available: <https://doi.org/10.1109/CSF.2017.11>
- [6] J. Dong, A. Roth, and W. J. Su, “Gaussian differential privacy,” *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2021.
- [7] D. Kifer, S. Messing, A. Roth, A. Thakurta, and D. Zhang, “Guidelines for implementing and auditing differentially private systems,” *CoRR*, vol. abs/2002.04049, 2020. [Online]. Available: <https://arxiv.org/abs/2002.04049>
- [8] M. Gaboardi, K. Nissim, and D. Purser, “The complexity of verifying loop-free programs as differentially private,” in *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, ser. Leibniz International Proceedings in Informatics, vol. 168, 2020.
- [9] M. Bun, M. Gaboardi, and L. Glinskih, “The complexity of verifying boolean programs as differentially private,” in *2022 IEEE 35th Computer Security Foundations Symposium*, 2022.
- [10] G. Barthe, M. Gaboardi, J. Hsu, and B. C. Pierce, “Programming language techniques for differential privacy,” *ACM SIGLOG News*, vol. 3, no. 1, pp. 34–53, 2016. [Online]. Available: <https://doi.org/10.1145/2893582.2893591>
- [11] J. Reed and B. C. Pierce, “Distance makes the types grow stronger,” in *ICFP’10: Proceedings of the 15th ACM SIGPLAN international conference on Functional programming*, 2010, pp. 157–168.
- [12] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce, “Linear dependent types for differential privacy,” in *POPL’13: Proceedings of the 40th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2013, pp. 357–370.
- [13] A. Azevedo de Amorim, M. Gaboardi, J. Hsu, and S. Katsumata, “Probabilistic relational reasoning via metrics,” in *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*. IEEE, 2019, pp. 1–19. [Online]. Available: <https://doi.org/10.1109/LICS.2019.8785715>
- [14] J. P. Near, D. Darais, C. Abua, T. Stevens, P. Gaddamadugu, L. Wang, N. Somani, M. Zhang, N. Sharma, A. Shan, and D. Song, “Duet: an expressive higher-order language and linear type system for statically enforcing

- differential privacy,” *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, 2019. [Online]. Available: <https://doi.org/10.1145/3360598>
- [15] M. Toro, D. Darais, C. Abuaah, J. P. Near, D. Árquez, F. Olmedo, and É. Tanter, “Contextual linear types for differential privacy,” *ACM Trans. Program. Lang. Syst.*, vol. 45, no. 2, pp. 8:1–8:69, 2023. [Online]. Available: <https://doi.org/10.1145/3589207>
- [16] V. Sannier and P. Baillot, “A linear type system for  $L^p$ -metric sensitivity analysis,” in *9th International Conference on Formal Structures for Computation and Deduction*, ser. Leibniz International Proceedings in Informatics, vol. 299, 2024.
- [17] j. wunder, A. Azevedo de Amorim, P. Baillot, and M. Gaboardi, “Bunched fuzz: Sensitivity for vector metrics,” in *Programming Languages and Systems: ESOP 2023*, T. Wies, Ed. Cham: Springer Nature Switzerland, Apr. 2023, pp. 451–478.
- [18] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum, “Differential privacy under continual observation,” in *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, L. J. Schulman, Ed. ACM, 2010, pp. 715–724.
- [19] M. Lyu, D. Su, and N. Li, “Understanding the sparse vector technique for differential privacy,” in *Proceedings of the VLDB Endowment*, vol. 10, no. 6, 2017, pp. 637–648.
- [20] M. Hardt and G. N. Rothblum, “A multiplicative weights mechanism for privacy-preserving data analysis,” in *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*. IEEE Computer Society, 2010, pp. 61–70.
- [21] S. Vadhan and W. Zhang, “Concurrent composition theorems for differential privacy,” in *STOC 2023: Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, 2023.
- [22] R. Milner, *Communicating and Mobile Systems: the  $\pi$ -Calculus*. Cambridge University Press, 1999.
- [23] R. Milner, J. Parrow, and D. Walker, “A calculus of mobile processes Pt. 1,” *Information and Computation*, vol. 100, no. 1, pp. 1–40, 1992.
- [24] K. Honda, V. T. Vasconcelos, and M. Kubo, “Language primitives and type discipline for structured communication-based programming,” in *Proceedings of ESOP*, ser. Lecture Notes in Computer Science, vol. 1381, 1998, pp. 122–138.
- [25] A. Ghosh, R. Roughgarden, and M. Sundararajan, “Universally utility-maximizing privacy mechanisms,” in *STOC’09: Proceedings of the forty-first annual ACM symposium on Theory of computing*. Association for Computing Machinery, 2009, pp. 351–360.
- [26] A. Azevedo de Amorim, M. Gaboardi, J. Hsu, S. ya Katsumata, and I. Cherigui, “A semantic account of metric preservation,” in *POPL’17: Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, ser. ACM SIGPLAN Notices, vol. 52, no. 1, 2017, pp. 545–556.
- [27] J.-Y. Girard, “Linear logic,” *Theoretical Computer Science*, vol. 50, no. 1, pp. 1–102, 1987.
- [28] F. D. McSherry, “Privacy integrated queries: An extensible platform for privacy-preserving data analysis,” in *SIGMOD’09: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009.
- [29] M. Dezani-Ciancaglini and U. De’Liguoro, “Sessions and session types: An overview,” in *WS-FM’09: Proceedings of the 6th international conference on Web services and formal methods*, 2009, pp. 1–28.
- [30] K. Honda, N. Yoshida, and M. Carbone, “Multiparty asynchronous session types,” *ACM SIGPLAN Notices*, vol. 43, no. 1, pp. 273–284, 2008.



- [31] N. Yoshida and V. T. Vasconcelos, “Language primitives and type discipline for structured communication-based programming revisited: Two systems for higher-order session communication,” in *Proceedings of the 1st International Workshop on Security and Rewriting Techniques (SecReT’06)*, ser. Electronic Notes in Theoretical Computer Science, vol. 171, no. 4. Elsevier, 2007, pp. 73–93.
- [32] C. Baier, B. Engelen, and M. Majster-Cederbaum, “Deciding bisimilarity and similarity for probabilistic processes,” *Journal of Computer and System Sciences*, vol. 60, no. 1, pp. 187–231, 2000.
- [33] K. Chatzikokolakis and C. Palamidessi, “A framework for analyzing probabilistic protocols and its application to the partial secrets exchange,” *Theoretical Computer Science*, vol. 389, no. 3, pp. 512–527, 2007.
- [34] G. Bian and A. Abate, “On the relationship between bisimulation and trace equivalence in an approximate probabilistic context,” in *Foundations of Software Science and Computation Structures*, ser. Lecture Notes in Computer Science, vol. 10203, 2017, pp. 321–337.
- [35] D. Winograd-Cort, A. Haeberlen, A. Roth, and B. C. Pierce, “A framework for adaptive differential privacy,” *Proc. ACM Program. Lang.*, vol. 1, no. ICFP, pp. 10:1–10:29, 2017. [Online]. Available: <https://doi.org/10.1145/3110254>
- [36] A. Das, D. Wang, and J. Hoffmann, “Probabilistic resource-aware session types,” *Proc. ACM Program. Lang.*, vol. 7, no. POPL, pp. 1925–1956, 2023. [Online]. Available: <https://doi.org/10.1145/3571259>
- [37] L. Xu, “Modular reasoning about differential privacy in a probabilistic process calculus,” in *Trustworthy Global Computing - 7th International Symposium, TGC 2012, Newcastle upon Tyne, UK, September 7-8, 2012, Revised Selected Papers*, ser. Lecture Notes in Computer Science, C. Palamidessi and M. D. Ryan, Eds., vol. 8191. Springer, 2012, pp. 198–212. [Online]. Available: [https://doi.org/10.1007/978-3-642-41157-1\\_13](https://doi.org/10.1007/978-3-642-41157-1_13)
- [38] M. C. Tschantz, D. K. Kaynar, and A. Datta, “Formal verification of differential privacy for interactive systems (extended abstract),” in *Twenty-seventh Conference on the Mathematical Foundations of Programming Semantics, MFPS 2011, Pittsburgh, PA, USA, May 25-28, 2011*, ser. Electronic Notes in Theoretical Computer Science, M. W. Mislove and J. Ouaknine, Eds., vol. 276. Elsevier, 2011, pp. 61–79. [Online]. Available: <https://doi.org/10.1016/j.entcs.2011.09.015>
- [39] D. Chistikov, A. S. Murawski, and D. Purser, “Bisimilarity distances for approximate differential privacy,” in *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, ser. Lecture Notes in Computer Science, S. K. Lahiri and C. Wang, Eds., vol. 11138. Springer, 2018, pp. 194–210. [Online]. Available: [https://doi.org/10.1007/978-3-030-01090-4\\_12](https://doi.org/10.1007/978-3-030-01090-4_12)
- [40] A. Evfimievski, J. Gehrke, and R. Srikant, “Limiting privacy breaches in privacy preserving data mining,” in *PODS’03: Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2003, pp. 211–222.
- [41] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith, “What can we learn privately?” in *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, 2008.