Session Types for the Concurrent Composition of Interactive Differential Privacy

Victor Sannier, Patrick Baillot, Marco Gaboardi

Session Types for the Concurrent Composition of Interactive Differential Privacy

Victor Sannier[1], Patrick Baillot[1], and Marco Gaboardi[2]

[1]Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL, F-59 000 Lille, France
[2]Department of Computer Science, Boston University, USA

December 21, 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 to the same database. An important part of these frameworks is the presence of composition theorems, which enable data curators to combine multiple differentially private algorithms, resulting in a new algorithm that still satisfies differential privacy. Deriving composition theorems within the concurrent interactive framework, as well as for advanced notions of DP, is a complex task, and some progress has been made in this area recently [1, 2]. On the other hand a variety of tools have been proposed for certifying that some given algorithms are DP. Among them, the typing approach embodied by the Fuzz language consists in using a functional programming language endowed with 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 therefore propose to extend it by using a process calculus similar to the π-calculus as the language. This calculus is equipped with operational semantics that enable us to express the DP property as an approximation of trace equivalence. Moreover, we introduce a type system in the form 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, (ϵ, δ)-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 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 (ϵ, δ)-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 cannot account for some increasingly common scenarios where the data analyst interacts with a database in a sequential adaptive way, and in particular for the use of primitives such as the Sparse Vector Technique (SVT) [18, 19], and the Private Multiplicative Weights [20].

Moreover, in some cases, a data analyst might want

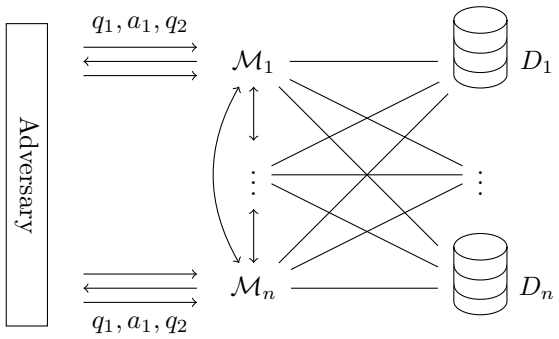


Figure 1: Concurrent composition of interactive mechanisms, where the adversary can arbitrarily interleave its queries

to perform multiple adaptive analyses concurrently and interleave 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].

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 π-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 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 an operational semantics based on probabilistic automata [25] to represent the behaviour of the system. Within this framework, we formalise what each process *sees* during an execution and how to quantitatively compare different executions. From this, we redefine interactive differential privacy, which was originally described by Vadhan and Wang [1] for abstract parties.

Finally, the third ingredient is a type system for this calculus that keeps track of the privacy parameters of the processes. Specifically, we will introduce a type system in the form of session types [24], an approach that has been thoroughly explored in the literature and offers a robust and versatile framework.

Once these three ingredients have been introduced, we will 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 differential privacy, expressed with some parameters given by its type. In particular, within our type system, the role of composition theorems will be fulfilled by the soundness of the typing rule for the concurrent composition of two or more mechanisms.

1.3 Contributions

In this article,

- We introduce a variant of the π-calculus with sessions that includes constructs for finite replication and random number generation, equipped with a probabilistic operational semantics;
- 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 (Non-Interactive) 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 (ϵ, δ)-differentially private if for all*

$X \subseteq \mathrm{Range}(\mathcal{M})$ *and for all adjacent inputs* x *and* x'*, we have*

$$\Pr[\mathcal{M}(x) \in X] \leq \mathrm{e}^{\epsilon} \Pr[\mathcal{M}(x') \in X] + \delta\,. \tag{1}$$

Remark 2.1 *Each application domain can use its own specific definitions for input and adjacency. However, the most common case is that two datasets are considered adjacent if one can transform one into the other by simply adding or removing a single 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*

$$d_Y\big(f(x), f(x')\big) \leq s \cdot d_X(x, x')\,. \tag{2}$$

Depending on the metric used in the output space to compute the sensitivity of the function, we may then add noise that is distributed according to different probability distributions.

Theorem 2.1 (Laplace mechanism [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*

$$x \mapsto (f_1(x) + Y_1, \ldots, f_n(x) + Y_n)\,, \tag{3}$$

where $Y_i \sim \mathcal{L}(s/\epsilon)$ *for all* i*, is* ϵ*-differentially private.*

Theorem 2.2 (Gaussian mechanism [4, Theorem 3.22]) *For all algorithms* f *mapping to* $\mathbf{R}^n$ *that are* s*-sensitive with respect to the* L^2 *metric, and for all privacy parameters* $\epsilon > 0$*, if* $\sigma^2 > 2\ln(1.25\delta/\delta)\, s^2/\epsilon^2$*, then the randomised algorithm*

$$x \mapsto (f_1(x) + Y_1, \ldots, f_n(x) + Y_n)\,, \tag{4}$$

where $Y_i \sim \mathcal{N}(0, \sigma^2)$ *for all* i*, is* (ϵ, δ)*-differentially private.*

Remark 2.2 *See Ghosh et al. [26] and Canonne et al. [27] for discrete versions of these mechanisms, using integers rather than real numbers.*

As a consequence of the above theorems, 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] have introduced Fuzz, a typed functional programming language where types are interpreted by metric spaces [28], and whose type constructors are derived from linear logic [29]. Thus, among other denotations,

- $A \otimes B$ is interpreted as the product space $[\![A]\!] \otimes [\![B]\!]$, endowed with the L^1-metric (that is, $d_{A\otimes B}\big((a,b),(a',b')\big) = d_A(a,a') + d_B(b,b')$),
- $[\![A \multimap B]\!]$ is the space of all 1-sensitive functions from $[\![A]\!]$ to $[\![B]\!]$, endowed with $d_{A\multimap B}(f,f') = \max_{x\in A} d_B\big(f(x), f(x')\big)$, and
- for all sensitivities s, $[\![!_s A]\!]$ is the space $[\![A]\!]$ 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 $[x_1 : A_1]_{s_1}, \ldots, [x_n : A_n]_{s_n} \vdash e : A$ and are sound when $[\![e]\!]$ is a 1-sensitive function from the product space $[\![!_{s_1} A_1]\!] \otimes \cdots \otimes [\![!_{s_n} A_n]\!]$ endowed with the L_1 distance to the metric space $[\![A]\!]$. They may also read, "For all i, $[\![e]\!]$ 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.

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

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

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

Finally, a monadic constructor $\bigcirc$ parametrised by a fixed ϵ ensures that if $\vdash f : A \multimap \bigcirc B$, then $[\![f]\!]$ is ϵ-differentially private from $[\![A]\!]$ to $[\![B]\!]$, 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 λ-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

The setting 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, 20]. As a reaction, a new notion of differential privacy has been introduced for interactive protocols Vadhan and Wang [1].

An interactive protocol is a pair of mutually recursive functions, referred to as *parties*, each with access to a random number generator and some input data. The view of a party is understood as "everything it sees during the execution" [1, Definition 1.6]. We write $\mathrm{View}(A \parallel M)$ for the view of A when interacting with M, and $\mathrm{View}(M \parallel A)$ for the view of M when interacting with A.

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

$$\Pr\big[\mathrm{View}\big(\mathcal{A} \parallel \mathcal{M}(x)\big) \in X\big] \leq \mathrm{e}^{\epsilon} \Pr\big[\mathrm{View}\big(\mathcal{A} \parallel \mathcal{M}(y)\big) \in X\big] + \delta\,. \quad (5)$$

Remark 2.3 *Note that we will use the term "adversary" throughout this paper, as we do not know the intentions of 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 2.3 ([30, Theorem 4]) *Let M_i be mechanisms, each providing ϵ-differential privacy. Let D_i be arbitrary disjoint subsets of the input domain D. The sequence of $M_i(X \cap D_i)$ provides ϵ-differential privacy.*

The second is concurrent composition, where two processes operate on the same data, potentially interleaving their message sending and receiving.

Definition 2.4 ([1, Definition 1.5]) *Let $M_1, \ldots, M_n$ be interactive mechanisms. Their concurrent composition $M = \mathrm{ConComp}(M_1, \ldots, M_n)$ is defined as follows:*

- *its random samples consist of $(r_1, \ldots, r_n)$, where r_i are the random samples for M_i,*
- *its inputs consist of $(x_1, \ldots, x_n)$, where x_i are the inputs for M_i,*
- *$M(x, m_0, \ldots, m_{i-1}; r)$ ouputs either 'halt' if m_{i-1} cannot be parsed as (i, q) where q is a query to M_i, or outputs $M_i(x_i, m_0^j, \ldots, m_{t-1}^i; r_i)$ where $(m_0^j, \ldots, m_{t-1}^j)$.*

We will present just one result from the literature on concurrent composition as an illustration, which is for (ϵ, δ)-differential privacy, where the privacy parameters for all mechanisms are equal.

Theorem 2.4 ([1, Theorem 1.8]) *If interactive mechanisms $M_1, \ldots, M_n$ are each (ϵ, δ)-differentially private, then their concurrent composition is $\big(k \cdot \epsilon, (\mathrm{e}^{k\epsilon} - 1)/(\mathrm{e}^{\epsilon} - 1) \cdot \delta\big)$-differentially private.*

Accordingly, we will later introduce two typing rules for process 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 π-calculus [22, 23], rather than functional languages like the λ-calculus. The core concept of the π-calculus is that of processes communicating over channels, enabling two-way interaction.

The syntax of the π-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 π-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 developed for this language, but we will concentrate on session types (introduced by Tekeuchi et al. [31] and extended by Honda et al. [24]; see Dezani-Ciancaglini and De'Liguoro [32] for an overview). These session types ensure that communication over a specified channel follows a particular sequence of actions, thereby 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 α" (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 α." 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 π-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 by the same lead author [33].

Remark 2.4 *We made two omissions from the usual syntax of the π-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 π-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 (ϵ, δ) within the typing judgements.

3.1 Server Replication and Random Number Generation

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

$$\begin{aligned} P, Q, \ldots ::= \cdots &\mid *_n k?(x) \,.\, P \\ &\mid \mathbf{Lap}_b?(x) \,.\, P \mid \mathbf{Gauss}_\sigma?(x) \,.\, P \end{aligned} \tag{6}$$

and accordingly the syntax of session types:

$$\alpha ::= \cdots \mid *_n?A \,.\, \alpha \,. \tag{7}$$

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

Remark 3.1 *Our constructs for generating random numbers resemble using the restriction operator ν in the applied π-calculus [34] for generating random keys and nonces. The main difference between the two approaches is that, in our context, the random numbers are used as noise and must be sampled from a specific probability distribution to apply Theorems 2.1 and 2.2, which we describe in our syntax. Furthermore, our constructs are not affected by structural rules.*

Remark 3.2 *We allow finite server 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 Γ 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 Γ is a classical context mapping expression variables and session names to types, written A and $\langle \alpha, \bar{\alpha} \rangle$ respectively, Δ is a linear context mapping session channels to session types and (ϵ, δ) are privacy parameters. Anticipating the semantics, the above a judgement reads, "Under the environment Γ and the typing Δ, P is an (ϵ, δ)-differentially private process."

We write $[\Gamma]_s$ for the Fuzz context obtained by adding a sensitivity annotation s to each expression type in the environment Γ and $\Gamma_1 \amalg \Gamma_2$ for the disjoint union of Γ_1 and Γ_2. Moreover, we say that a typing Δ is *completed* when it only contains end types [35, Section 2.2].

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 Γ 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:*

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

Figure 2: Syntax of the π-calculus with session types

- *if $P \equiv_\alpha Q$ (if P is α-equivalent to Q), then $P \equiv Q$,*
- $*_0 P \equiv 0$,
- $(\nu u)0 \equiv 0$, $(\nu u u')P \equiv (\nu u' u)P$, *and if* $u \notin \operatorname{fu} Q \cup \operatorname{fu} Q$, *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 3.3 *Contrary to Honda et al. [24, Section 2.3], both Honda et al. [33, Figure 2] and Yoshida and Vasconcelos [35, Figure 2] consider that, in general, $(\nu u u)P$ and $(\nu u)P$ are not congruent processes. We adhere to the latter works.*

Most π-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 4.7 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:

$$(\epsilon_1, \delta_1) \star (\epsilon_2, \delta_2) = (\epsilon_1 + \epsilon_2, \delta_1 + \mathrm{e}^{\epsilon_1}\delta_2) \,. \tag{8}$$

Observe that this operation is neither commutative nor associative.

Remark 3.4 *In order to obtain tighter bounds, we could 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, each process performs a symmetrical role to the other. For each message received by one process, there is a corresponding message sent by the other process, 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 α is defined by structural induction using the following equations: 1. $\bar{\bar{\alpha}} = \alpha$, 2. $\overline{?A \,.\, \alpha} = !A \,.\, \bar{\alpha}$, 3. $\overline{\&_i\{l_i : \alpha_i\}} = \oplus_i\{l_i : \bar{\alpha}_i\}$, 4. $\overline{*_n \alpha} = *_n \bar{\alpha}$.*

Definition 3.3 ([24, Definition 5.2]) *We say that two typings Δ_1 and Δ_2, which map session channels to session types, are* compatible *whenever for all channels k in $\operatorname{dom}(\Delta_1) \cap \operatorname{dom}(\Delta_2)$, we have $\Delta_1(k) = \overline{\Delta_2(k)}$. 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 Δ_1 and Δ_2 is*

defined by

$$(\Delta_1 \circ \Delta_2)(k) = \begin{cases} \bot & \text{if } k \in \mathrm{dom}(\Delta_1) \cap \mathrm{dom}(\Delta_2) \\ \Delta_i(k) & \text{if } k \in \mathrm{dom}(\Delta_i) \setminus \mathrm{dom}(\Delta_{3-i}) \end{cases} \quad (9)$$

3.4 Typing Rules

The typing rules for our process calculus are 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 [35, 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-Lap] for the Laplace mechanism;
- [T-NSend-Gauss] for the Gaussian mechanism;
- [T-Conc] for concurrent composition;
- [T-Par] for parallel composition; and
- [T-Rep] for 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 [35, 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.

Note that this type system can be straightforwardly extended to incorporate, for example, the Laplace and Gaussian mechanisms over vectors (where the L^1 and L^2 vector metrics no longer coincide) rather than integers. This can be achieved by embedding Plurimetric Fuzz, as described by Sannier and Baillot [16], which analyses sensitivity according to vector metrics, as our expression language, instead of using Fuzz [11].

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

$$M = k?(f) \,.\, \mathbf{Lap}_{1/\epsilon}?(r) \,.\, k![f(D) + r] \,.\, \mathrm{end}$$
$$A = k![f] \,.\, k?(y) \,.\, 0 \,.$$

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

$$D : \mathrm{Data}, f : \mathrm{Data} \multimap \mathrm{Int} \vdash M \triangleright k : \alpha; (\epsilon, 0)$$
$$f : \mathrm{Data} \multimap \mathrm{Int} \vdash A \triangleright k : \bar{\alpha}; (0, 1)$$

Example 3.2 *A variation of Example 3.1 would be to use the Gaussian mechanism instead of the Laplace mechanism. For all (ϵ, δ) and all σ such that $\sigma^2 > 2/\epsilon^2 \ln(5/4\delta)$, we can derive the typing judgement*

$$D : \mathrm{Data}, f : \mathrm{Data} \multimap \mathrm{Int} \vdash M \triangleright k : \alpha; (\epsilon, \delta)$$

for the following mechanism

$$M = k?(f) \,.\, \mathbf{Gauss}_\sigma?(r) \,.\, k![f(D) + r] \,.\, \mathrm{end} \,.$$

where α is of the same session type as above.

(To obtain closed programs from the above examples, we would just need to abstract the channels using the [T-Acc] and [T-Req] typing rules, and thus add a session name to the environment.)

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 datasets, 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 generally ensure differential privacy. The data could indeed be private (it might even consist of the entire dataset), or it could leak private information. This situation occurs, for example, when the noise r, which is used to mask the output y of a function operating on a dataset, is revealed (as knowing both y and $y + r$ is sufficient to infer the value of y.) Therefore, in order to achieve meaningful privacy parameters, it is necessary to use a rule like [T-NSend], which requires the process to first add newly generated 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 (Theorem 5.8) that allows one to effectively use the [T-Par] rule with different privacy parameters in its premises or the [T-Conc] rule with some variables used in only one of the composed processes.

4 Operational Semantics

4.1 Reduction Rules

We will provide an operational semantics for our process calculus, presented as a probabilistic automaton (introduced by Segala and Lynch [25, Section 3.1] and simplified by Chatzikokolakis and Palamidessi [36, Section 2.2]), the labels of which will trace the execution. In the next section, we will use these labels to define the view of a party during an interaction.

$$\frac{\Delta \text{ completed}}{\Gamma \vdash 0 \triangleright \Delta; (\epsilon, \delta)}\ [\text{T-Inact}] \qquad \frac{\Gamma \vdash P \triangleright \Delta, k : \text{end}; (\epsilon, \delta)}{\Gamma \vdash P \triangleright \Delta, k : \bot; (\epsilon, \delta)}\ [\text{T-Bot}]$$

$$\frac{\Gamma \vdash a : \langle \alpha, \bar{\alpha} \rangle \quad \Gamma \vdash P \triangleright \Delta, k : \alpha; (\epsilon, \delta)}{\Gamma \vdash a(k) \,.\, P \triangleright \Delta; (\epsilon, \delta)}\ [\text{T-Acc}] \qquad \frac{\Gamma \vdash a : \langle \alpha, \bar{\alpha} \rangle \quad \Gamma \vdash P \triangleright \Delta, k : \bar{\alpha}; (\epsilon, \delta)}{\Gamma \vdash \bar{a}(k) \,.\, P \triangleright \Delta; (\epsilon, \delta)}\ [\text{T-Req}]$$

$$\frac{[\Gamma]_{+\infty} \vdash_{\text{Fuzz}} e : A \quad \Gamma \vdash P \triangleright \Delta, k : \alpha; (\epsilon, \delta)}{\Gamma \vdash k![e] \,.\, P \triangleright \Delta, k : !A \,.\, \alpha; (0, 1)}\ [\text{T-Send}] \qquad \frac{\Gamma, x : A \vdash P \triangleright \Delta, k : \alpha; (\epsilon, \delta)}{\Gamma \vdash k?(x) \,.\, P \triangleright \Delta, k : ?A \,.\, \alpha; (\epsilon, \delta)}\ [\text{T-Rcv}]$$

$$\frac{[\Gamma]_s \vdash_{\text{Fuzz}} e : \text{Int} \quad \Gamma \vdash P \triangleright \Delta, k : \alpha; (\epsilon, \delta)}{\Gamma \vdash \mathbf{Lap}_{s/\epsilon'}?(x) \,.\, k![e + x] \,.\, P \triangleright \Delta, k : !\text{Int} \,.\, \alpha; (\epsilon + \epsilon', \delta)}\ [\text{T-NSend-Lap}]$$

$$\frac{[\Gamma]_s \vdash_{\text{Fuzz}} e : \text{Int} \quad \Gamma \vdash P \triangleright \Delta, k : \alpha; (\epsilon, \delta) \quad \sigma^2 > 2\ln(1.25/\delta')\, s^2/\epsilon'^2}{\Gamma \vdash \mathbf{Gauss}_\sigma?(x) \,.\, k![e + x] \,.\, P \triangleright \Delta, k : !\text{Int} \,.\, \alpha; (\epsilon + \epsilon', \delta + \delta')}\ [\text{T-NSend-Gauss}]$$

$$\frac{\Gamma, x : \text{Int} \vdash P \triangleright \Delta; (\epsilon, \delta) \quad k \in \{\mathbf{Lap}_b, \mathbf{Gauss}_\sigma\}_{b,\sigma>0}}{\Gamma \vdash k?(x) \,.\, P \triangleright \Delta; (\epsilon, \delta)}\ [\text{T-Rand}]$$

$$\frac{[\Gamma]_s \vdash_{\text{Fuzz}} e : \text{Bool} \quad \Gamma \vdash P \triangleright \Delta; (\epsilon_P, \delta_P) \quad \Gamma \vdash Q \triangleright \Delta; (\epsilon_Q, \delta_Q)}{\Gamma \vdash \textbf{if } e \textbf{ then } P \textbf{ else } Q \triangleright \Delta; (0, 1)}\ [\text{T-If}]$$

$$\frac{\Gamma \vdash P_1 \triangleright \Delta, k : \alpha_1; (\epsilon, \delta) \quad \ldots \quad \Gamma \vdash P_n \triangleright \Delta, k : \alpha_n; (\epsilon, \delta)}{\Gamma \vdash k \triangleright \{l_i : P_i\}_{i \in I} \triangleright \Delta, k : \&_{i \in I}\{l_i : \alpha_i\}; (\epsilon, \delta)}\ [\text{T-Br}]$$

$$\frac{\Gamma \vdash P \triangleright \Delta, k : \alpha_j; (\epsilon, \delta) \quad j \in I}{\Gamma \vdash k \triangleleft l_j \,.\, P \triangleright \Delta, k : \oplus_{i \in I}\{l_i : \alpha_i\}; (\epsilon, \delta)}\ [\text{T-Sel}_j]$$

$$\frac{\Gamma \vdash P_1 \triangleright \Delta_1; (\epsilon_1, \delta_1) \quad \Gamma \vdash P_2 \triangleright \Delta_2; (\epsilon_2, \delta_2) \quad \Delta_1 \asymp \Delta_2}{\Gamma \vdash P_1 \parallel P_2 \triangleright \Delta_1 \circ \Delta_2; (\epsilon_1, \delta_1) \star (\epsilon_2, \delta_2)}\ [\text{T-Conc}]$$

$$\frac{\Gamma_1 \vdash P_1 \triangleright \Delta_1; (\epsilon, 0) \quad \Gamma_2 \vdash P_2 \triangleright \Delta_2; (\epsilon, 0) \quad \Delta_1 \asymp \Delta_2}{\Gamma_1 \amalg \Gamma_2 \vdash P_1 \parallel P_2 \triangleright \Delta_1 \circ \Delta_2; (\epsilon, 0)}\ [\text{T-Par}]$$

$$\frac{\Gamma, a : \langle \alpha, \bar{\alpha} \rangle \vdash P \triangleright \Delta; (\epsilon, \delta)}{\Gamma \vdash (\nu a) P \triangleright \Delta; (\epsilon, \delta)}\ [\text{T-NHide}] \qquad \frac{\Gamma \vdash P \triangleright \Delta, k : \bot;\ (\epsilon, \delta)}{\Gamma \vdash (\nu k) P \triangleright \Delta; (\epsilon, \delta)}\ [\text{T-CHide}]$$

$$\frac{\Gamma \vdash P \triangleright \Delta; (\epsilon, \delta) \quad P \equiv Q}{\Gamma \vdash Q \triangleright \Delta; (\epsilon, \delta)}\ [\text{T-Cong}] \qquad \frac{\Gamma \vdash k?(x) \,.\, P \triangleright \Delta, k : ?A \,.\, \alpha; (\epsilon, \delta) \quad \vdash_{\text{Fuzz}} n : \text{Nat}}{\Gamma \vdash *_n k?(x) \,.\, P \triangleright \Delta, k : *_n ?A \,.\, \alpha; \left(n\epsilon, \delta \frac{e^{n\epsilon} - 1}{e^\epsilon - 1}\right)}\ [\text{T-Rep}]$$

Figure 3: Process Typing Rules

Definition 4.1 *Given a set S of states and a set A of actions (also known as labels), a* probabilistic automaton *is defined as a triple (S, T, s_0) where $s_0 \in S$ is an initial state, and $T \subseteq S \times \mathrm{Dist}(A \times S)$ is a set of* transition groups.

The central idea is that the selection between transition groups is done non-deterministically, whereas the selection of a transition within a group is made probabilistically. In particular, when there is a single transition group for every initial state, we say that the reduction is fully probabilistic.

To write the rules for probabilistic automata, we use the notation from the process calculus π_{prob} [36, Figure 1]. Specifically, we write

$$P\left\{\xrightarrow[p_i]{l_i} Q_i\right\}_{i \in I}$$

when for all i in I, a process P reduces to Q_i with probability p_i and label l_i, meaning that $\big(P, (\{(l_i, Q_i)\}_i, p)\big)$ is a transition group.

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, ϵ_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 subscript tuple when it is equal to 1.) Atoms can be concatenated using the + operator 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] and [R-Gauss], as expected, are the only rules that introduce randomness, and they do so through the discrete Laplace distribution and discrete normal distribution, respectively.

Definition 4.2 (Ghosh et al. [26, Example 2.1]) *The discrete Laplace distribution with parameter $b > 0$, is defined by the following probability mass function over $\mathbf{Z}$:*

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

Definition 4.3 (Canonne et al. [27, Definition 1]) *The discrete normal distribution with parameter $\sigma > 0$, is defined by the following probability mass function over $\mathbf{Z}$:*

$$n \mapsto \underbrace{\frac{1}{\sum_{n \in \mathbf{Z}} \mathrm{e}^{-n^2/2\sigma^2}} \times \mathrm{e}^{-n^2/2\sigma^2}}_{q_{n,\sigma}} . \tag{11}$$

The other reduction rules merely propagate this randomness.

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.

Multi-step reduction

Lemma 4.1 *If $P\left\{\xrightarrow[p_i]{l_i} Q_i\right\}_i$, and for all i we have $Q_i\left\{\xrightarrow[p'_j]{l'_j} Q'_j\right\}_j$, then for all i and j, the trees l_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. □

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

Definition 4.4 *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 $\big(a, (b, c)\big)$ and $\big(d, (e, [f, g])\big)$ is the tree $\big([a, d], ([b, e], [c, f, g])\big)$.*

Definition 4.5 (Two-step reduction of a fully probabilistic au
For any two processes P and R, let X_{PR} be the set of all processes Q such that P can reduce to Q 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 Q such that $l_{PQ} + l_{QR} = l$ with a non-zero probability. 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

$$P\left\{\xrightarrow[p_i]{l_i}{}^2 R_i\right\}$$

to account for all such R.

Starting from a process P, given a function to resolve the nondeterminism (that is a *scheduler* [36, Section 2.2]), the resulting automaton becomes fully probabilistic. Consequently, it is associated with a fully probabilistic two-step reduction (see Definition 4.5), which corresponds to a single transition group.

Definition 4.6 (Two-step reduction of a probabilistic automa
The two-step reduction for the non-deterministic probabilistic automaton generated by P is defined as the union over all possible schedulers of the associated transition groups.

Note that two transition groups obtained by two different schedulers can, in fact, be the same. This occurs when the diamond property is satisfied for the associated one-step reductions.

$$\frac{e \downarrow \top}{\textbf{if } e \textbf{ then } P \textbf{ else } Q \left\{ \xrightarrow[1]{\epsilon_\top} P \right\}} \text{ [R-True]} \qquad \frac{e \downarrow \bot}{\textbf{if } e \textbf{ then } P \textbf{ else } Q \left\{ \xrightarrow[1]{\epsilon_\bot} P \right\}} \text{ [R-False]}$$

$$\frac{e \downarrow v}{k![e] \,.\, P \parallel k?(x) \,.\, Q \left\{ \xrightarrow[1]{(\alpha_{1,v}, \alpha_{1,v})} P \parallel Q[v/x] \right\}} \text{ [R-Val]} \qquad \frac{}{a(k) \,.\, P \parallel \bar{a}(k) \,.\, Q \left\{ \xrightarrow[1]{(\chi_k, \chi_k)} (\nu k)(P \parallel Q) \right\}} \text{ [R-Chan]}$$

$$\frac{}{k \triangleleft l_i \,.\, P \parallel k \triangleright \{l_i : P_i\}_i \left\{ \xrightarrow[1]{(\delta_i, \delta_i)} P \parallel P_i \right\}} \text{ [R-Sel]}$$

$$\frac{P \left\{ \xrightarrow[p_i]{l_i} P_i \right\}_i}{(\nu u) P \left\{ \xrightarrow[p_i]{l_i} (\nu u) P_i \right\}_i} \text{ [R-Hide]} \qquad \frac{P \left\{ \xrightarrow[p_i]{l_i} P_i \right\}_i}{P \parallel Q \left\{ \xrightarrow[p_i]{(l_i, \emptyset)} P_i \parallel Q \right\}_i} \text{ [R-Conc]}$$

$$\frac{}{\textbf{Lap}_b?(x) \,.\, P \left\{ \xrightarrow[p_{n,b}]{\gamma_{L,b,n}} P[n/x] \right\}_{n \in \mathbf{Z}}} \text{ [R-Lap]} \qquad \frac{}{\textbf{Gauss}_\sigma?(x) \,.\, P \left\{ \xrightarrow[q_{n,\sigma}]{\gamma_{G,\sigma,n}} P[n/x] \right\}_{n \in \mathbf{Z}}} \text{ [R-Gauss]}$$

$$\frac{e \downarrow v}{k![e] \,.\, P_1 \parallel *_n k?(x) \,.\, P_2 \left\{ \xrightarrow[1]{(\alpha_{n,v}, \alpha_{n,v})} P_1 \parallel (P_2[v/x] \parallel *_{n-1} k?(x) \,.\, P_2) \right\}} \text{ [R-Rep]}$$

$$\frac{P' \left\{ \xrightarrow[p_i]{l_i} Q'_i \right\}_i \qquad P \equiv P' \qquad (\forall i)(Q_i \equiv Q'_i)}{P \left\{ \xrightarrow[p_i]{l_i} Q_i \right\}_i} \text{ [R-Cong]}$$

$$\frac{P_1 \parallel P_2 \left\{ \xrightarrow[p_i]{(l_i, r_i)} Q_{i_1} \parallel Q_{i_2} \right\}_i}{P_2 \parallel P_1 \left\{ \xrightarrow[p_i]{(r_i, l_i)} Q_{i_2} \parallel Q_{i_1} \right\}_i} \text{ [R-Comm]} \qquad \frac{P_1 \parallel (P_2 \parallel P_3) \left\{ \xrightarrow[p_i]{(l_1, (l_2, l_3))} P_{1i} \parallel (P_{2i} \parallel P_{3i}) \right\}_i}{(P_1 \parallel P_2) \parallel P_3 \left\{ \xrightarrow[p_i]{((l_1, l_2), l_3)} (P_{1i} \parallel P_{2i}) \parallel P_{3i} \right\}_i} \text{ [R-Assoc]}$$

Figure 4: Process Reduction Rules

Example 4.2 *Let P such that $P\{\xrightarrow[1/2]{l_i} P_i\}_{i\in\{1,2\}}$ and Q such that $Q\{\xrightarrow[1]{r} R\}$. The process $P \parallel Q$ can be reduced according to two transition groups in one step, that is*

$$P \parallel Q\Big\{\xrightarrow[1/2]{(l_i,\emptyset)} P_i \parallel Q\Big\}_{i\in\{1,2\}} \quad \text{and} \quad P \parallel Q\Big\{\xrightarrow[1]{(\emptyset,r)} P \parallel R\Big\}$$

(the latter is obtaiend by applying [R-Conc] followed by [R-Comm]), but according to only one group in two steps, that is $P \parallel Q\Big\{\xrightarrow[1/2]{(l_i,r)}{}^{2} P_i \parallel R\Big\}_{i\in\{1,2\}}$.

Lemma 4.2 *For all processes P, all numbers k, and every transition group $\big(P, (\{(l_i, Q_i)\}_i, p)\big)$ of the probabilistic automaton with the initial state P, which is generated by k iterations of the rules presented in Figure 4, it holds that $\sum_i p_i = 1$.*

Lemma 4.3 *For every process P, there exists a number n such that the probabilistic automaton with initial state P, generated by n iterations of the rules in Figure 4, has a single transition group and all the processes it reaches are in normal form[1].*

Proof. The π-calculus with session types enjoys the strong normalisation property [37]. Moreover, all normalisation paths have the same length. □

From the two lemmas above, we can deduce the existence of a normalisation relation.

Theorem 4.4 *There exists a fully probabilistic normalization, denoted by $\Rightarrow$, that is associated with the reduction rules of Figure 4.*

Traces We then call a possible label for this normalisation 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 Ω, the trace $\mathrm{Tr}(P)$ of the execution of a process P, where

$$P\Big\{\overset{l_i}{\underset{p_i}{\Longrightarrow}} P_i\Big\}_i,$$

is the naturally defined random variable such that for all i

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

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

Lemma 4.5 *For all processes P and Q, the probability that $\mathrm{Tr}(P \parallel Q)$ 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. □

[1] Recall that a process is in normal form if it cannot be reduced any further.

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

Definition 4.7 *The* view of a process A interacting with a process M, written as $\mathrm{View}(A \parallel M)$, *is the following random variable:* $\mathrm{Left}\big(\mathrm{Tr}(A \parallel M)\big)$, *that is,* $\omega \mapsto \mathrm{Left}\big(\mathrm{Tr}(A \parallel M)(\omega)\big)$.

Remark 4.1 *Except in rare particular cases, we have* $\mathrm{View}(A \parallel M) \neq \mathrm{View}(M \parallel A)$.

Example 4.3 *Taking the same example as before (Example 3.1), for every integer n,*

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

where $l_n = [\alpha_f, \alpha_{f(D)+n}]$, and $r_n = [\alpha_f, \gamma_{L,1/\epsilon,n}, \alpha_{f(D)+n}]$. In addition, for all n, we have $\Pr[\mathrm{View}(A \parallel M) = l_n] = p_{n,1/\epsilon}$ and $\Pr[\mathrm{View}(M \parallel A) = r_n] = p_{n,1/\epsilon}$. 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 $\gamma_{L,1/\epsilon,n}$ in its view, the adversary A never has access to the generated random value n. This prevents them from substracting n from the final result $f(D) + n$ to infer the private information $f(D)$.

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 4.8 *Given an environment Γ and a typing Δ, we say that a process M is (ϵ, δ)-*differentially private *if for all adjacent substitutions S and S' that are well-typed for Γ, for all adversary processes A such that $A \parallel M[S]$ (and $A \parallel M[S']$) are closed processes, and all $X \subseteq \mathrm{Range}(\mathcal{M})$, we have the following inequality:*

$$\Pr\big[\mathrm{View}(A \parallel M[S]) \in X\big] \leq \mathrm{e}^{\epsilon} \Pr\big[\mathrm{View}(A \parallel M[S']) \in X\big] + \delta\,. \quad (12)$$

In practice, S and S' will often be substitutions involving a single variable, which represents the entire dataset, and the adjacency of two substitutions will correspond to the adjacency of their associated datasets. However, this relation is subject to redefinition depending on the specific area of application.

Remark 4.2 *Bian and Abate [38, 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 4.8); 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*

$$\begin{aligned}\Pr\big[\mathrm{Tr}\big((P \parallel Q) \parallel R\big) = \big((t_1, t_2), t_3\big)\big] \\ = \Pr\big[\mathrm{Tr}\big(P \parallel (Q \parallel R)\big) = \big(t_1, (t_2, t_3)\big)\big]\,. \quad (13)\end{aligned}$$

Proof. This is a consequence of the structural reduction rule [R-Assoc]. □

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

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

$$\begin{aligned}\Pr\big[\mathrm{View}\big((P \parallel Q) \parallel R\big) \in \mathrm{Left}^*(T)\big] \\ = \Pr\big[\mathrm{View}\big(P \parallel (Q \parallel R)\big) \in T\big]\,. \quad (14)\end{aligned}$$

Proof.

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

Note that we use Lemma 5.1 to prove the third equality. □

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,

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

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

$$\begin{aligned}&\Pr\big[\mathrm{View}\big(A \parallel (M_1[S] \parallel M_2[S])\big) \in T\big] \\ &\quad \le \mathrm{e}^{\epsilon_1} \Pr\big[\mathrm{View}\big(A \parallel (M_1[S'] \parallel M_2[S])\big) \in T\big]\,.\end{aligned}$$

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

$$\begin{aligned}&\Pr\big[\mathrm{View}\big(A \parallel (M_1[S] \parallel M_2[S])\big) \in T\big] \\ &= \Pr\big[\mathrm{View}\big((A \parallel M_1[S]) \parallel M_2[S]\big) \in \mathrm{Left}^*(T)\big] \\ &\le \mathrm{e}^{\epsilon_2} \Pr\big[\mathrm{View}\big((A \parallel M_1[S]) \parallel M_2[S']\big) \in \mathrm{Left}^*(T)\big] \\ &= \mathrm{e}^{\epsilon_2} \Pr\big[\mathrm{View}\big(A \parallel (M_1[S] \parallel M_2[S'])\big) \in T\big]\end{aligned}$$

The inequality arises from our hypothesis that M_2 is an ϵ_2-differentially private process. □

Lemma 5.4 *The [T-NSend-Lap], [T-NSend-Gauss], and [T-Par] typing rules are sound.*

Proof. See Theorems 2.1 to 2.3. □

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

Proof. This result follows from Vadhan and Wang [1, Theorem 1.8]. The authors achieve improved privacy parameters compared to what we would obtain by repeatedly applying the [T-Conc] rule, as discussed in Section 3.3.2. □

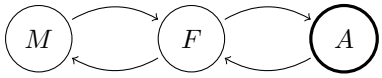
Finally, we obtain the following soundness theorem: if a process M is well-typed, then it is (ϵ, δ)-differentially private (as defined in Definition 4.8), and discloses only a limited amount of private information when interacting with *any* adversary A that meets the given assumptions.

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

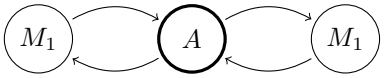
Note that even though our semantics differ, a process written in the fragment of our language that is common to Honda et al. is reducible according to their semantics if and only if it is reducible according to ours. As a consequence, we obtain the same good properties for our language as Honda et al. [24, Theorem 5.4].

Theorem 5.7

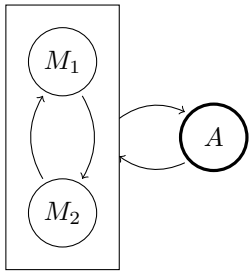
- *Typing is preserved by reduction:*



(a) Example 1



(b) Example 2



(c) Example 3

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

If $\Gamma \vdash P \triangleright \Delta; (\epsilon, \delta)$ *and* $P\left\{\frac{l_i}{p_i} \to^* Q_i\right\}$, *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 [32, Section 2].

Moreover, as previously mentioned, one can freely add variables to the typing contexts or increase the privacy parameters.

Theorem 5.8 (Weakening) *If* $x \notin \Gamma$, $a \notin \Gamma$, *and* $\Gamma \vdash P \triangleright \Delta; (\epsilon, \delta)$, *then* $\Gamma, x : A, a : \langle \alpha, \bar{\alpha} \rangle \vdash P \triangleright \Delta; (\epsilon + \epsilon', \delta + \delta')$.

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 Data to Int, a dataset D, and channels k and k' of session type $\alpha = ?(\text{Data} \multimap \text{Int}) . !\text{Int} . \text{end}$, 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 $A[f] \| (F \| M[D])$, the adversary A, and indeed every possible well-typed adversary, 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 ϵ-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 dataset. Let us also consider a data analyst A who wants to approximately estimate the total number of entries by summing up the results obtained from querying M_1 and M_2 using the count function. By applying [T-Par], we get

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

for the straightforward session types α and β.

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

Let 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 dataset 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

$\rho \leftarrow \mathcal{L}(1/\epsilon)$;
for $i = 1, 2, \ldots, T$ **do**
  Receive the next query (f_i, τ_i);
  $\gamma_i \leftarrow \mathcal{L}(c/\epsilon)$;
  if $|f_i(X) - \tau_i| + \gamma_i \geq E + \rho$ **then**
    $v_i \leftarrow f(X) + \mathcal{L}(c/\epsilon)$;
    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 dataset 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
```

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

$$\frac{}{\Gamma \vdash \text{SVT}(c, E, N, D, k, a) \triangleright \Delta; (3\epsilon, 0)} \text{ [T-SVT]} \quad (15)$$

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

Remark 6.2 *Even though we can write the sparse vector technique using our language, which eliminates the need to introduce additional reduction rules, we are unable to derive the above typing judgement from the rules shown in Figure 3 because verifying its soundness requires probabilistic reasoning. Therefore, we consider the sparse vector technique (SVT) as a primitive and accept this typing judgement as a typing rule, a common decision in this context [11].*

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

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

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

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. [39], it consists in a two-layer language and does not deal with any concurrency aspects.

Concerning probabilistic versions of the π-calculus, Chatzikokolakis and Palamidessi [36] defined the π_{prob}-calculus, a variant of π-calculus with probabilistic choice,

and Das et al. [40] described a language for probabilistic session types, and similarly to π_{prob} uses a construction for probabilistic branching. However none of these two papers is motivated by differential privacy. Our approach is unique because we substitute the probabilistic choice with a construct for generating random numbers. This is crucial for our type system, as it allows us to manage non-trivial (ϵ, δ) bounds, which would not be directly feasible with probabilistic choice.

Xu [41] 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 or Gaussian mechanisms. On the other hand, Ding et al. [42] consider a variant of the π-calculus, not for differential privacy but rather for location privacy (which involves trying to conceal the exact location of a user).

Other works [43, 44, 45] connect differential privacy to trace distances or bisimilarity distances for probabilistic labelled transition systems. However, their approach is quite different from ours, as they do not consider a process calculus as we do in this paper, nor do they address the concurrent composition of interactive differential privacy.

8 Future Work

First, we would like to investigate whether we could employ more refined probabilistic reasoning to derive well-known mechanisms, such as the sparse vector technique (see Section 6.3), rather than treating them as primitives.

Then, we aim to determine whether our process calculus and its associated type system can be applied to other concepts of differential privacy.

- If future research were to explore the generation of an infinite sequence of random numbers instead of just a finite one (see Section 3.1) thus proposing a new definition of a private process, and if they proved composition theorems for this new definition, we could replace our replication construct with recursive processes. This would raise interesting semantic issues, particularly in relation to measure theory, since the probability spaces involved would have the cardinality of the continuum.

- We would also like to determine if we could ensure privacy properties in different contexts, particularly within the field of local differential privacy [46, 47], where there is no central aggregator; instead, each party possesses a portion of the private data.

Finally, as outlined in Section 4.3, 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.

Acknowledgment

This work was partially funded by the ANR Project HOPR (ANR-24-CE48-5521-01) from the French National Research Agency and by a research grant from the Région Hauts-de-France.

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.

[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.

[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.

[14] J. P. Near, D. Darais, C. Abuah, 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.

[15] M. Toro, D. Darais, C. Abuah, 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.

[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 π-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] R. Segala and N. Lynch, "Probabilistic simulations for probabilistic processes," in *CONCUR'94: Concurrency Theory*, ser. Lecture Notes in Computer Science, vol. 836, 1994, pp. 481–496.

[26] 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.

[27] C. L. Canonne, G. Kamath, and T. Steinke, "The discrete gaussian for differential privacy," in *NIPS'20: Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020, pp. 15 676–15 688.

[28] 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.

[29] J.-Y. Girard, "Linear logic," *Theoretical Computer Science*, vol. 50, no. 1, pp. 1–102, 1987.

[30] 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.

[31] K. Tekeuchi, K. Honda, and M. Kubo, "An interaction-based language and its typing system," in *PARLE'94: Proceedings of the 6th International PARLE Conference on Parallel Architectures and Languages Europe*, 1994, pp. 398–413.

[32] 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.

[33] K. Honda, N. Yoshida, and M. Carbone, "Multiparty asynchronous session types," *ACM SIGPLAN Notices*, vol. 43, no. 1, pp. 273–284, 2008.

[34] M. Abadi, B. Blanchet, and C. Fournet, "The applied pi calculus: Mobile values, new names, and secure communication," *Journal of the ACM*, vol. 65, no. 1, pp. 1–41, 2017.

[35] 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.

[36] 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.

[37] J. A. Pérez, L. Caires, F. Pfenning, and B. Toninho, "Linear logical relations and observational equivalences for session-based concurrency," in *Information and Computation*, vol. 239, 2014, pp. 254–302.

[38] 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.

[39] 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.

[40] A. Das, D. Wang, and J. Hoffmann, "Probabilistic resource-aware session types," *Proc. ACM Program. Lang.*, vol. 7, no. POPL, pp. 1925–1956, 2023.

[41] 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.

[42] J. Ding, X. Li, Y. Guo, L. Yin, and H. Zhang, "Process calculus for modeling and quantifying location privacy," *Procedia Computer Science*, vol. 147, pp. 407–415, 2009.

[43] 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.

[44] 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.

[45] V. Castiglioni, K. Chatzikokolakis, and C. Palamidessi, "A logical characterization of differential privacy," *Science of Computer Programming*, vol. 188, 2020.

[46] 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.

[47] 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.