# A Typing System for Privacy

Dimitrios Kouzapas[1(✉)] and Anna Philippou[2]

[1] Department of Computing, Imperial College London, London, UK
dk208@doc.ic.ac.uk
[2] Department of Computer Science, University of Cyprus, Nicosia, Cyprus
annap@cs.ucy.ac.cy

**Abstract.** In this paper we report on work-in-progress towards defining a formal framework for studying privacy. Our framework is based on the $\pi$-calculus with groups [1] accompanied by a type system for capturing privacy-related notions. The typing system we propose combines a number of concepts from the literature: it includes the use of *groups* to enable reasoning about information collection, it builds on *read/write capabilities* to control information processing, and it employs *type linearity* to restrict information dissemination. We illustrate the use of our typing system via simple examples.

## 1 Introduction

The notion of privacy does not have a single solid definition. It is generally viewed as a collection of related rights as opposed to a single concept and attempts towards its formalization have been intertwined with philosophy, legal systems, and society in general. The ongoing advances of network and information technology introduce new concerns on the matter of privacy. The formation of large databases that aggregate sensitive information of citizens, the exchange of information through e-commerce as well as the rise of social networks, impose new challenges for protecting individuals from violation of their right to privacy as well as for providing solid foundations for understanding privacy a term.

A study of the diverse types of privacy, their interplay with technology, and the need for formal methodologies for understanding and protecting privacy is discussed in [7], where the authors base their arguments on the taxonomy of privacy rights by Solove [6]. According to [6], the possible privacy violations within a system can be categorized into four groups: *invasions*, *information collection*, *information processing*, and *information dissemination*. These violations are typically expressed within a model consisting of three entities: the *data subject* about whom a *data holder* has information and the *environment*, the data holder being responsible to protect the information of the data subject against unauthorized adversaries in the environment.

The motivation for this work stems from the need to provide a formal framework (or a set of different formal frameworks) for reasoning about privacy-related concepts, as discussed above. Such a framework would provide solid foundations

S. Counsell and M. Núñez (Eds.): SEFM 2013 Collocated Workshops, LNCS 8368, pp. 56–68, 2014.
DOI: 10.1007/978-3-319-05032-4_5, 

for understanding the notion privacy and it would allow to rigorously model and study privacy-related situations. Our interest for formal privacy is primarily focused on the processes of *information collection*, *information processing*, and *information dissemination* and how these can be controlled in order to guarantee the preservation of privacy within a system.

### 1.1 Privacy and the $\pi$-Calculus

The approach we follow in this paper attempts to give a correspondence between the requirements of the last paragraph and the theory and meta-theory of the $\pi$-calculus [4]. The $\pi$-calculus is a formal model of concurrent computation that uses message-passing communication as the primitive computational function. A rich theory of operational, behavioural and type system semantics of the $\pi$-calculus is used as a tool for the specification and the study of concurrent systems. Our aim is to use the $\pi$-calculus machinery to describe notions of privacy. Specifically, we are interested in the development of a meta-theory, via a typing system, for the $\pi$-calculus that can enforce properties of privacy, as discussed above.

The semantics for the $G\pi$-calculus, a $\pi$-calculus that disallows the leakage of information (secrets) is presented in [1]. That work proposes the *group type* along with a simple typing system that is used to restrict the scope of a name's existence, i.e., a name cannot exist outside its group scope. We find the semantics of the $G\pi$-calculus convenient to achieve the privacy properties regarding the *information collection* category. A data holder can use the group type to disallow unauthorized adversaries from collecting information about a data subject.

Consider for example the processes:

$$\begin{aligned} \mathsf{DBadmin} &= \overline{a}\langle c\rangle.\mathbf{0} \\ \mathsf{Nurse} &= a(x).\overline{b}\langle x\rangle.\mathbf{0} \\ \mathsf{Doctor} &= b(x).x(y).\overline{x}\langle \mathsf{data}\rangle.\mathbf{0} \end{aligned}$$

The database administrator process DBadmin sends a reference $c$ to a patient's data to a doctor process Doctor using a nurse process Nurse as a delegate. Channel $c$ is sent to the nurse via channel $a$ and is then forwarded to the doctor via channel $b$ by the nurse. The doctor then uses $c$ to read and write data on the patient's records. The composition of the above processes under the fresh hospital group Hosp, and an appropriate typing of $c$, enforces that no external adversary will be able to collect the information exchanged in the above scenario, namely $c$: name $c$, belonging to group Hosp, is not possible to be leaked outside the defined context because (1) groups are not values and cannot be communicated and (2) the group Hosp is only known by the three processes (see [1] for the details).

$$(\nu\ \mathsf{Hosp})(((\nu c : \mathsf{Hosp}[])\mathsf{DBadmin}) \mid \mathsf{Nurse} \mid \mathsf{Doctor})$$

Let us now move on to the concept of *information processing* and re-consider the example above with the additional requirement that the nurse should not

be able to read or write on the patient's record in contrast to the doctor who is allowed both of these capabilities. To address this issue we turn to the input/output typing system for the $\pi$-calculus of Pierce and Sangiorgi, [5]. Therein, the input/output subtyping is used to control the input and output capabilities on names and it is a prime candidate for achieving privacy with respect to the requirement in question: A type system that controls read and write capabilities[1] can be used by a data holder to control how the information about a data subject can be processed. Thus, in the case of our example, the requirements may be fulfilled by extending the specification with a read/write typing system as follows:

$$\begin{aligned} T_{\mathsf{data}} &= \mathsf{Hosp}[\mathsf{MedicalData}]^{-} \\ T_c &= \mathsf{Hosp}[T_{\mathsf{data}}]^{\mathtt{rw}} \\ T_a &= \mathsf{Hosp}[\mathsf{Hosp}[T_{\mathsf{data}}]^{-}]^{\mathtt{rw}} \\ T_b &= \mathsf{Hosp}[\mathsf{Hosp}[T_{\mathsf{data}}]^{\mathtt{rw}}]^{\mathtt{rw}} \end{aligned}$$

where names $a$, $b$ and $c$ are of types $T_a$, $T_b$ and $T_c$, respectively. The medical data are a basic type with no capability of read and write. Channel $c$ can be used for reading and writing medical data. Channel $a$ is used to pass information to the nurse without giving permission to the nurse to process the received information, while channel $b$ provides read and write capabilities to the doctor. Nonetheless, the above system suffers from the following problem. Although the nurse acquires restricted capabilities for channel $c$ via channel $a$, it is still possible for a nurse process to exercise its read capability on $b$ and, thus, acquire read and write capability on the $c$ channel. To avoid this problem, the system may be redefined as follows:

$$\begin{aligned} \mathsf{DBadmin} &= (\nu b : T_b)\ \overline{tonurse}\langle b\rangle.\overline{todoc}\langle b\rangle.\overline{a}\langle c\rangle.\mathbf{0} \\ \mathsf{Nurse} &= tonurse(z).a(x).\overline{z}\langle x\rangle.\mathbf{0} \\ \mathsf{Doctor} &= todoc(z).z(x).x(y).\overline{x}\langle \mathsf{data}\rangle.\mathbf{0} \end{aligned}$$

where channel *todoc* has type $\mathsf{Hosp}[T_b]^{rw}$ but channel *tonurse* has type $\mathsf{Hosp}[T'_b]^{rw}$, where $T'_b = \mathsf{Hosp}[\mathsf{Hosp}[T_{\mathsf{data}}]^{\mathtt{rw}}]^{\mathtt{w}}$. In other words, the nurse is not assigned read capabilities on channel $b$.

Note that the above typing is not completely sound: for instance the nurse process is expected to pass on to the doctor process more capabilities than those it acquires via channel $a$. Nevertheless in our theory we use a more complex type structure able to solve this problem.

Regarding the *information dissemination* category of privacy violations, we propose to handle information as a *linear resource*. Linear resources are resources that can be used for some specific number of times. A typing system for linearity was originally proposed in [3]. A linear typing system can be used by the data holder to control the number of times an information can be disseminated. In our

[1] The terminology for read and write capabilities is equivalent with input and output terminology.

example, we require from the nurse the capability of sending the reference of the patient only once, while we require from the doctor not to share the information with anyone else:

$$\begin{aligned} T_{\mathsf{data}} &= \mathsf{Hosp}[\mathsf{MedicalData}]^{-*} \\ T_c &= \mathsf{Hosp}[T_{\mathsf{data}}]^{\mathtt{rw}*} \\ T_a &= \mathsf{Hosp}[\mathsf{Hosp}[T_{\mathsf{data}}]^{-1}]^{\mathtt{rw}0} \\ T_b &= \mathsf{Hosp}[\mathsf{Hosp}[T_{\mathsf{data}}]^{\mathtt{rw}0}]^{\mathtt{rw}0} \\ T'_b &= \mathsf{Hosp}[\mathsf{Hosp}[T_{\mathsf{data}}]^{\mathtt{rw}0}]^{\mathtt{w}0} \end{aligned}$$

The $*$ annotation on the types above defines a shared (or unlimited) resource. Such resources are the patient's data and the reference to the patient's data. Channels $a$ and $b$ communicate values that can be disseminated one and zero times respectively. (Again there is a soundness problem solved by a more complex typing structure.) Furthermore channels $a$ and $b$ cannot be sent to other entities.

A central aspect of our theory is the distinction between the basic entities. The operational semantics of the $\pi$-calculus focuses on the communication between processes that are composed in parallel. Although a process can be thought of as a computational entity, it is difficult to distinguish at the operational level which processes constitute a logical entity. In our approach, we do not require any operational distinction between entities, since this would compromise the above basic intuition for the $\pi$-calculus, but we do require the logical distinction between the different entities that compose a system.

Finally, we note that our typing system employs a combination of i/o types and linear types, which are low-level $\pi$-calculus types, to express restrictions on system behavior. We point out that the expressivity of such ordinary $\pi$-calculus types has been studied in the literature and, for instance, in [2] the authors in fact prove that linear and variant types can be used to encode session types.

## 2 The Calculus

Our study of privacy is based on the $\pi$-calculus with groups proposed by Cardelli et al. [1]. In this section we briefly overview the syntax and reduction semantics of the calculus.

Beginning with the syntax, this is standard $\pi$-calculus syntax with the addition of the group restriction construct, $(\nu\ G)P$, and the requirement for typing on bound names (the definition of types is in Sect. 3).

$$P \ ::= x(y{:}T).P \ \mid\ \overline{x}\langle z\rangle.P \ \mid\ (\nu\ G)P \ \mid\ (\nu\ a{:}T)P \ \mid\ P_1 \mid P_2 \ \mid\ !P \ \mid\ \mathbf{0}$$

Free names $\mathtt{fn}(P)$, bound names $\mathtt{bn}(P)$, free variables $\mathtt{fv}(P)$, and bound variables $\mathtt{bv}(P)$ are defined in the standard way for $\pi$-calculus processes. We extend this notion to the sets of free groups in a process $P$ and a type $T$ which we denote as $\mathtt{fg}(P)$ and $\mathtt{fg}(T)$, respectively.

We now turn to defining the reduction semantics of the calculus. This employs the notion of *structural congruence* which allows the structural rearrangement of a process so that the reduction rules can be performed. Structural congruence is the least congruence relation, written $\equiv$, that satisfies the rules:

$$
\begin{array}{ll}
P \mid \mathbf{0} \equiv P & (\nu\ a{:}T)P_1 \mid P_2 \equiv (\nu\ a : T)(P_1 \mid P_2) \text{ if } a \notin \mathtt{fn}(P_2) \\
P_1 \mid P_2 \equiv P_2 \mid P_1 & (\nu\ a{:}T_1)(\nu\ b{:}T_2)P \equiv (\nu\ b{:}T_2)(\nu\ a{:}T_1)P \\
(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3) & (\nu\ G)P_1 \mid P_2 \equiv (\nu\ G)(P_1 \mid P_2) \text{ if } G \notin \mathtt{fg}(P_2) \\
!P \equiv P \mid !P & (\nu\ G_1)(\nu\ G_2)P \equiv (\nu\ G_2)(\nu\ G_1)P \\
\multicolumn{2}{c}{(\nu\ G_1)(\nu\ a{:}T)P \equiv (\nu\ a{:}T)(\nu\ G_1)P \text{ if } G \notin \mathtt{fg}(T)}
\end{array}
$$

We may now present the reduction relation $P \longrightarrow Q$ which consists of the standard $\pi$-calculus reduction relation extended with a new rule for group creation.

$$
\begin{array}{rll}
\multicolumn{3}{c}{\overline{a}\langle b\rangle.P_1 \mid a(x : T).P_2 \longrightarrow P_1 \mid P_2\{b/x\}} \\
P_1 \longrightarrow P_2 & \text{implies} & P_1 \mid P_3 \longrightarrow P_2 \mid P_3 \\
P_1 \longrightarrow P_2 & \text{implies} & (\nu\ G)P_1 \longrightarrow (\nu\ G)P_2 \\
P_1 \longrightarrow P_2 & \text{implies} & (\nu\ a : T)P_1 \longrightarrow (\nu\ a : T)P_2 \\
P_1 \equiv P_1', P_1' \longrightarrow P_2', P_2' \equiv P_2 & \text{implies} & P_1 \longrightarrow P_2
\end{array}
$$

## 3 Types and Typing System

In this section we define a typing system for the calculus which builds upon the typing of [1]. The typing system includes: (i) the notion of groups of [1], (ii) the read/write capabilities of [5] extended with the empty capability, and (iii) a notion of linearity on the dissemination of names. The type structure is used for static control over the permissions and the disseminations on names in a process.

For each channel, its type specifies (1) the group it belongs to, (2) the type of values that can be exchanged on the channel, (3) the ways in which the channel may be used in input/output positions (permissions $p$ below) and (4) the number of times it may be disseminated (linearity $\lambda$ below):

$$
\begin{array}{rcll}
T & ::= & G[]^{p\lambda} \mid G[T]^{p\lambda} & \\
p & ::= & - \mid \mathtt{r} \mid \mathtt{w} \mid \mathtt{rw} & \\
\lambda & ::= & * \mid i & \text{where } i \geq 0
\end{array}
$$

For example, a channel of type $T = G[]^{\mathtt{r}\,2}$ is a channel belonging to group $G$ that does not communicate any names, can be used in input position and twice

in object position. Similarly, a name of type $G'[T]^{\mathtt{rw}*}$ is a channel of group $G'$ that can be used in input and output position for exchanging names of type $T$ and can be sent as the object of a communication for an arbitrary number of times.

**Subtyping.** Our typing system makes use of a subtyping relation which, in turn is, based on two pre-orders, one for permissions $p$, denoted as $\sqsubseteq_p$, and one for linearities $\lambda$, denoted as $\sqsubseteq_\lambda$:

$$\begin{array}{llll} \sqsubseteq_p: & \mathtt{rw} \sqsubseteq_p \mathtt{w} & \mathtt{rw} \sqsubseteq_p \mathtt{r} & \mathtt{rw}, \mathtt{r}, \mathtt{w} \sqsubseteq_p - \\ \sqsubseteq_\lambda: & * \sqsubseteq_\lambda i \quad \text{for all } i & i \sqsubseteq_\lambda j \quad \text{if } i \geq j & \end{array}$$

The preorder for permissions is as expected with the empty capability being the greatest element. For linearities, *fewer* permissions are included in *larger* permissions and $*$ is the least element.

Let $\mathsf{Type}$ be the set of all types $T$. The subtyping relation, written $\leq$ as an infix notation, may be defined coinductively as the largest fixed point ($\mathscr{F}^\omega(\mathsf{Type}\times \mathsf{Type})$) of the monotone function:

$$\mathscr{F} : (\mathsf{Type} \times \mathsf{Type}) \longrightarrow (\mathsf{Type} \times \mathsf{Type})$$

where

$$\begin{array}{l} \mathscr{F}(\mathscr{R}) = \{(G[]^{-0}, G[]^{-0})\} \\ \quad \cup \{(G[T_1]^{p\lambda_1}, G[T_2]^{-\lambda_2})) \mid (T_1, T_2) \in \mathscr{R}, (T_2, T_1) \in \mathscr{R}, \lambda_1 \sqsubseteq_\lambda \lambda_2\} \\ \quad \cup \{(G[T_1]^{p\lambda_1}, G[T_2]^{\mathtt{r}\lambda_2}) \mid (T_1, T_2) \in \mathscr{R}, p \sqsubseteq_p \mathtt{r}, \lambda_1 \sqsubseteq_\lambda \lambda_2\} \\ \quad \cup \{(G[T_1]^{p\lambda_1}, G[T_2]^{\mathtt{w}\lambda_2}) \mid (T_2, T_1) \in \mathscr{R}, p \sqsubseteq_p \mathtt{w}, \lambda_1 \sqsubseteq_\lambda \lambda_2\} \\ \quad \cup \{(G[T_1]^{\mathtt{rw}\lambda_1}, G[T_2]^{\mathtt{rw}\lambda_2}) \mid (T_1, T_2), (T_2, T_1) \in \mathscr{R}, \lambda_1 \sqsubseteq_\lambda \lambda_2\} \end{array}$$

The first pair in the construction of $\mathscr{F}$ says that the least base type is reflexive. The next four cases define subtyping based on the preorders defined for permissions and linearities. According to the second case, the empty permission is associated with an invariant subtyping relation because the empty permission disallows for a name to be used for reading and/or writing. The read permission follows covariant subtyping, the write permission follows contravariant subtyping, while the read/write permission follows invariant subtyping. Note that linearities are required to respect the relation $\lambda_1 \sqsubseteq \lambda_2$ for subtyping in all cases. For example, according to the subtyping relation, the following hold: $G_1[G_2[]^{\mathtt{rw}5}]^{\mathtt{rw}*} \leq G_1[G_2[]^{\mathtt{w}3}]^{\mathtt{r}3}$, $G_1[G_2[]^{-3}]^{\mathtt{rw}*} \leq G_1[G_2[]^{\mathtt{w}3}]^{\mathtt{w}0}$, and $G_1[G_2[]^{\mathtt{w}5}]^{\mathtt{rw}*} \leq G_1[G_2[]^{\mathtt{w}5}]^{-1}$.

**Typing Judgements.** We now turn to the typing system of our calculus. This assigns an extended notion of a type on names which is constructed as follows:

$$\mathsf{T} = (T_1, T_2)$$

In a pair $\mathsf{T}$ we record the current capabilities of a name, captured by $T_1$, and its future capabilities after its dissemination, captured by $T_2$.

Based on these extended types, the environment on which type checking is carried out in our calculus consists of the components $\Pi$ and $\Gamma$. These declare the names (free and bound) and groups in scope during type checking. We define $\Gamma$-environments by $\Gamma ::= \emptyset \mid \Gamma \cdot x : \mathsf{T} \mid \Gamma \cdot G$. The domain of an environment $\Gamma$, $\mathtt{dom}(\Gamma)$, is considered to contain all names and groups recorded in $\Gamma$. We assume that any name and group in $\mathtt{dom}(\Gamma)$ occurs exactly once in $\Gamma$. Then, a $\Pi$-environment is defined by $\Pi ::= \emptyset \mid \Pi \cdot x : \mathsf{T}$, where $\mathtt{dom}(\Pi)$ contains all variables in $\Pi$, each of which must exist in $\Pi$ exactly once.

We define three typing judgements: $\Gamma \vdash x \triangleright T$, $\Pi \vdash x \triangleright T$, and $\Pi, \Gamma \vdash P$. The first two typing judgement say that under the typing environment $\Gamma$, respectively $\Pi$, variable $x$ has type $T$. The third typing judgement stipulates that process $P$ is well typed under the environments $\Pi, \Gamma$, where $\Gamma$ records the groups and the types of the free names of $P$ and $\Pi$ the types of all bound names $x$ that are created via a $(\nu x)$ construct within $P$. We require that these bound names are uniquely named within $P$ and, if needed, we employ $\alpha$ conversion to achieve this. In essence, this restriction requires for all freshly-created names to be recorded a-priori within the typing environment. If an unrecorded name is encountered, then the typing system will lead to failure as is implemented by the typing system. It turns out that recording this information on bound names of a process is necessary in order to control the internal processing of names that carry sensitive data.

**Typing System.** We now move on to the rules of our typing system. First, we present two auxiliary functions. To begin with we define the linearity addition operator $\oplus$ where $\lambda_1 \oplus \lambda_2 = *$, if $\lambda_1 = *$ or $\lambda_2 = *$, and $\lambda_1 \oplus \lambda_2 = \lambda_1 + \lambda_2$, otherwise. We may now lift this notion to the level of typing environments via operator $\odot$ which composes its arguments by concatenating their declarations with the exception of the common domain where linearities are added up via $\oplus$:

$$\begin{array}{rcl} \Gamma_1 \odot \Gamma_2 & = & \Gamma_1 \backslash \Gamma_2 \cdot \Gamma_2 \backslash \Gamma_1 \\ & \cdot & \{x : G[T]^{p\lambda_1 \oplus \lambda_2} \mid x : G[T]^{p\lambda_1} \in \Gamma_1, x : G[T]^{p\lambda_2} \in \Gamma_2\} \end{array}$$

At this point we make the implicit assumption that $\Gamma_1$ and $\Gamma_2$ are compatible in the sense that the declared types of common names may differ only in the linearity component.

We are ready now define the typing system:

$$\text{(Name)}\ \frac{x \notin \mathtt{dom}(\Gamma \cdot \Gamma') \quad \mathtt{fg}(\mathsf{T}) \subseteq \mathtt{dom}(\Gamma \cdot \Gamma')}{\Gamma \cdot x : \mathsf{T} \cdot \Gamma' \vdash x \triangleright \mathsf{T}} \qquad \text{(SubN)}\ \frac{\Gamma \vdash x \triangleright (T_1', T_2'), T_1' \leq T_1, T_2' \leq T_2}{\Gamma \vdash x \triangleright (T_1, T_2)}$$

$$\text{(In)}\ \frac{\Pi, \Gamma \cdot y : (T_1, T_2) \vdash P \quad \Gamma \vdash x \triangleright (G[T_1]^{\mathtt{r0}}, G[T_2]^{\mathtt{r0}})}{\Pi, \Gamma \vdash x(y : T_1).P} \qquad \text{(Out)}\ \frac{\Pi, \Gamma \cdot y : (G_y[T_1]^{-\lambda}, T_2) \vdash P \quad \Gamma \vdash x \triangleright (G_x[T_2]^{\mathtt{w0}}, G_x[T_2]^{\mathtt{w0}})}{\Pi, \Gamma \cdot y : (G_y[T_1]^{-(\lambda \oplus 1)}, T_2) \vdash \overline{x}\langle y \rangle.P}$$

$$\text{(ResG)}\ \frac{\Pi, \Gamma \cdot G \vdash P}{\Pi, \Gamma \vdash (\nu\ G)P} \qquad \text{(ResN)}\ \frac{\Pi, \Gamma \cdot x : (T, T') \vdash P}{\Pi \cdot x : (T, T'), \Gamma \vdash (\nu\ x : T)P}$$

$$\text{(Par)}\ \frac{\Pi_1, \Gamma_1 \vdash P_1 \quad \Pi_2, \Gamma_2 \vdash P_2}{\Pi_1 \odot \Pi_2, \Gamma_1 \odot \Gamma_2 \vdash P_1 \mid P_2} \qquad \text{(Rep)}\ \frac{\begin{array}{c}\Pi, \Gamma \vdash P \\ \forall x \in \mathtt{fn}(P)\ \text{if}\ \Gamma \vdash x \triangleright (G[T_1]^{p\lambda_1}, G[T_2]^{p\lambda_2}) \\ \text{then}\ \lambda_1 \in \{0, *\}\end{array}}{\Pi, \Gamma \vdash !P}$$

$$\text{(Nil)}\ \ \Pi, \Gamma \vdash \mathbf{0} \qquad \text{(SubP)}\ \frac{\Pi, \Gamma \cdot x : (T_1', T_2') \vdash P \quad T_1' \leq T_1, T_2' \leq T_2}{\Pi, \Gamma \cdot x : (T_1, T_2) \vdash P}$$

Rule (Name) is used to type names. Note that in name typing we require that all group names of the type are present in the typing environment. Rule (SubN) defines a subsumption based on subtyping for channels. Rule (In) types the input prefixed process. We first require that the input subject has at least permission for reading. Then, the type $y$ is included in the type environment $\Gamma$ with a type that matches the type of the input channel $x$. This is to ensure that the input object will be used as specified. The rule for the output prefix (Out) checks that the output subject has write permissions. Furthermore, $x$ should be a channel that can communicate names up-to type $T_2$, the maximum type by which $y$ can be disseminated. Then, the continuation of the process $P$, should be typed according to the original type of $y$ and with its linearity reduced by one. Finally, the output object should have at least the empty permission.

In rule (ResG) we record a newly-created name in $\Gamma$. For name restriction (ResN) specifies that a process type checks only if the restricted name is recorded in environment $\Pi$. In this way is is possible to control the internal behavior of a process, in order to avoid possible privacy violations. Parallel composition uses the $\odot$ operator to compose typing environments, since we want to add up the linearity usage of each name. For the replication operator, axiom (Rep) we require that free names of $P$ have either linearity zero (i.e. they are not sent by $P$) or infinite linearity (i.e. they can be sent as many times as needed). The inactive process can be typed under any typing environment (axiom (Nil)). Finally we have a subsumption rule, (SubP) that uses subtyping to control the permissions on processes.

**Type Soundness.** We prove that the typing system is sound through a subject reduction theorem. Before we proceed with the subject reduction theorem we state the basic auxiliary lemmas.

**Lemma 1 (Weakening).**

1. If $\Gamma \vdash x \triangleright \mathsf{T}$ and $y \notin \mathtt{dom}(\Gamma)$ then $\Gamma \cdot y : \mathsf{T}' \vdash x \triangleright \mathsf{T}$.
2. If $\Pi, \Gamma \vdash P$ and $y \notin \mathtt{dom}(\Gamma)$ then $\Pi, \Gamma \cdot y : \mathsf{T} \vdash P$.

**Lemma 2 (Strengthening).**

1. If $\Gamma \cdot y : \mathsf{T}' \vdash x \triangleright \mathsf{T}$, $y \neq x$, then $\Gamma \vdash x \triangleright \mathsf{T}$.
2. If $\Pi, \Gamma \cdot y : \mathsf{T} \vdash P$ and $y \notin \mathtt{fn}(P)$ then $\Pi, \Gamma \vdash P$.

**Lemma 3 (Substitution).** If $\Pi, \Gamma \cdot x : \mathsf{T} \vdash P$ and $\Gamma \vdash y \triangleright \mathsf{T}$ then $\Pi, \Gamma \vdash P\{y/x\}$

**Lemma 4 (Subject Congruence).** If $\Pi, \Gamma \vdash P_1$ and $P_1 \equiv P_2$ then $\Pi, \Gamma \vdash P_2$.

We are now ready to state the Subject Reduction theorem.

**Theorem 1 (Subject Reduction).** Let $\Pi, \Gamma \vdash P$ and $P \longrightarrow P'$ then $\Pi, \Gamma \vdash P'$.

*Proof.* The proof is by induction on the reduction structure of $P$.
Basic Step:

$P = \overline{a}\langle b \rangle.P_1 \mid a(x).P_2 \longrightarrow P_1 \mid P_2\{b/x\}$ and $\Pi, \Gamma \vdash P$. From the typing system we get that

$$\Gamma = \Gamma_1 \odot \Gamma_2 \tag{1}$$
$$\Pi_1, \Gamma_1 \vdash \overline{a}\langle b \rangle.P_1 \tag{2}$$
$$\Pi_2, \Gamma_2 \vdash a(x).P_2 \tag{3}$$

From the typing system we get that $\Pi_1, \Gamma_1 \vdash P_1$ for (2) and $\Pi_2, \Gamma_2 \cdot x : T \vdash P_2$ for (3). We apply the substitution lemma (Lemma 3) to get that $\Pi_2, \Gamma_2 \cdot b : T \vdash P_2\{b/x\}$. We can now conclude that $\Pi, \Gamma \vdash P_1 \mid P_2\{b/x\}$.
Induction Step:

*Case: Parallel Composition.* Let $P_1 \mid P_2 \longrightarrow P_1' \mid P_2$ with $\Pi, \Gamma \vdash P_1 \mid P_2$. From the induction hypothesis we know that $\Pi_1, \Gamma_1 \vdash P_1$ and $\Pi_1, \Gamma_1 \vdash P_1'$. From these two results and the parallel composition typing we can conclude that $\Pi_1 \odot \Pi_2, \Gamma_1 \odot \Gamma_2 \vdash P_1 \mid P_2$ and $\Pi_1 \odot \Pi_2, \Gamma_1 \odot \Gamma_2 \vdash P_1' \mid P_2$ as required.

*Case: Group Restriction.* Let $(\nu\ G)P \longrightarrow (\nu\ G)P'$ with $\Pi, \Gamma \vdash (\nu\ G)P$. From the induction hypothesis we know that $\Pi, \Gamma \cdot G \vdash P$ and $\Pi, \Gamma \cdot G \vdash P'$. If we apply the name restriction rule on the last result we get $\Pi, \Gamma \vdash (\nu\ G)P'$.

*Case: Name Restriction.* Let $(\nu\ a : T)P \longrightarrow (\nu\ a : T)P'$ with $\Pi \cdot a : T, \Gamma \vdash P$. From the induction hypothesis we know that $\Pi, \Gamma \cdot a : T \vdash P$ and $\Pi, \Gamma \cdot a : T \vdash P'$. If we apply the name restriction rule on the last result we get $\Pi \cdot a : T, \Gamma \vdash (\nu\ a : T)P'$.

*Case: Structural Congruence Closure.* We use the subject congruence lemma (Lemma 4).

Let $P = P_1, P_1 \longrightarrow P_2, P_2 \cong P'$ with $\Pi, \Gamma \vdash P$. We apply subject congruence on $P$ to get $\Pi, \Gamma \vdash P_1$. Then we apply the induction hypothesis and subject congruence once more to get the required result. □

# 4 Examples

In this section we show simple use cases that apply the theory developed. We also show how we tackle different problems that might arise.

## 4.1 Patient Privacy

Our first example revisits our example from the introduction and completes the associated type system. Recall the scenario where a database administrator (process $\mathsf{DBadmin}$) sends a reference to the medical data of a patient to a doctor (process $\mathsf{Doctor}$) using a nurse (process $\mathsf{Nurse}$) as a delegate.

$$\begin{aligned}\mathsf{DBadmin} &= (\nu b : T_b)\ \overline{\mathsf{tonurse}}\langle b\rangle.\overline{\mathsf{todoc}}\langle b\rangle.\overline{a}\langle c\rangle.\mathbf{0}\\ \mathsf{Nurse} &= \mathsf{tonurse}(z).a(x).\overline{z}\langle x\rangle.\mathbf{0}\\ \mathsf{Doctor} &= \mathsf{todoc}(z).z(x).x(y).\overline{x}\langle\mathsf{data}\rangle.\mathbf{0}\end{aligned}$$

The processes are composed together inside the hospital ($\mathsf{Hosp}$) group.

$$\mathsf{Hospital} = (\nu\ \mathsf{Hosp})(((\nu c : T_c)\mathsf{DBadmin})\mid \mathsf{Nurse}\mid \mathsf{Doctor})$$

Our prime interest is to avoid leakage of the data during their dissemination to the doctor. This means that the nurse should not have access to the patient's data. On the other hand the doctor should be able to read and update medical data, but not be able to send the data to anyone else. We can control the above permissions using the following typing.

We define the types

$$\begin{aligned}\mathsf{T}_{\mathsf{data}} &= \mathsf{Hosp}[]^{-*}\\ \mathsf{T}_c &= \mathsf{Hosp}[\mathsf{T}_{\mathsf{data}}]^{\mathtt{rw}*}\\ \mathsf{T}_a &= \mathsf{Hosp}[\mathsf{Hosp}[\mathsf{T}_{\mathsf{data}}]^{-1}]^{\mathtt{rw}0}\\ \mathsf{T}'_a &= \mathsf{Hosp}[\mathsf{T}_c]^{\mathtt{rw}0}\\ \mathsf{T}_b &= \mathsf{Hosp}[\mathsf{Hosp}[\mathsf{T}_{\mathsf{data}}]^{\mathtt{rw}0}]^{\mathtt{rw}2}\\ \mathsf{T}^n_b &= \mathsf{Hosp}[\mathsf{Hosp}[\mathsf{T}_{\mathsf{data}}]^{\mathtt{rw}0}]^{\mathtt{w}0}\\ \mathsf{T}^d_b &= \mathsf{Hosp}[\mathsf{Hosp}[\mathsf{T}_{\mathsf{data}}]^{\mathtt{rw}0}]^{\mathtt{r}0}\\ \mathsf{T}_{td} &= \mathsf{Hosp}[\mathsf{T}^n_b]^{\mathtt{rw}0}\\ \mathsf{T}_{tn} &= \mathsf{Hosp}[\mathsf{T}^d_b]^{\mathtt{rw}0}\end{aligned}$$

to construct:

$$\begin{aligned}D &= (\mathsf{T}_{\mathsf{data}}, \mathsf{T}_{\mathsf{data}})\\ C &= (\mathsf{T}_c, \mathsf{T}_c)\\ A &= (\mathsf{T}_a, \mathsf{T}'_a)\\ B &= (\mathsf{T}_b, \mathsf{T}_b)\\ TD &= (\mathsf{T}_{td}, \mathsf{T}_{td})\\ TN &= (\mathsf{T}_{tn}, \mathsf{T}_{tn}).\end{aligned}$$

We can show that:

$$b : B \cdot c : C, \mathsf{tonurse} : TN \cdot \mathsf{todoc} : TD \cdot a : A \cdot \mathsf{data} : D \vdash \mathsf{Hospital}$$

Now, let us consider the case where the nurse sends channel $c$ on a private channel in an attempt to gain access on the patient's medical data:

$$\mathsf{Nurse}_2 \;=\; \mathsf{tonurse}(z).a(x).(\nu\ e : T_b)(\overline{e}\langle x\rangle.\mathbf{0} \mid e(y).y(w).\mathbf{0})$$

In this case, in order for the resulting system to type-check, the type of name $e$ would be recorded in the environment $\Pi$, as in

$$e : B \cdot b : B \cdot c : C, \mathsf{tonurse} : TN \cdot \mathsf{todoc} : TD \cdot a : A \cdot \mathsf{data} : D \vdash (\nu\ \mathsf{Hosp})(((\nu c : T_c)\mathsf{DBadmin}) \mid \mathsf{Nurse}_2 \mid \mathsf{Doctor})$$

This implies that, if we allow the creation of $e$, there is possibility of violation in a well-typed process. To avoid this, the administrator of the system should observe all names created and included in $\Pi$ and, in this specific case, disallow the creation of $e$. In future work we intend to address this point by providing typing policies that capture this type of problems and to refine our type system to disallow such privacy violations, possibly by controlling the process of name creation.

### 4.2 Social Network Privacy

Social networks allow users to share information within social groups. In the example that follows we define a type system to control the privacy requirements of participating users. In particular, we consider the problem where a user can make a piece of information public (e.g. a picture), but require that only specific people (his friends) can see it (and do nothing else with it).

The example considers a user who makes public the address, paddr, of a private object, pic, and wishes only the friend Friend to be able to read pic through the public address paddr. To achieve this the user makes available through the typing of name public only the object capability for paddr. However, by separately providing the friend with name $a$, it is possible to extend the capabilities of paddr to the read capability. In this way, channel $a$ acts as a key for Friend to unlock this private information. Assuming that notAFriend does not gain access to a name of type $T_a$, as in the process below, he will never be able to obtain read capability on channel paddr.

$$\begin{aligned}
\mathsf{User} &= (\nu a : T_a)(\overline{\mathsf{tofriend}}\langle a\rangle.(\nu \mathsf{paddr} : T_{paddr})(!\overline{\mathsf{public}}\langle \mathsf{paddr}\rangle.\mathbf{0} \mid !\overline{\mathsf{paddr}}\langle \mathsf{pic}\rangle.\mathbf{0})) \\
\mathsf{notAFriend} &= \mathsf{public}(z).\mathbf{0} \\
\mathsf{Friend} &= \mathsf{tofriend}(x).\mathsf{public}(y).(\overline{x}\langle y\rangle.\mathbf{0} \mid x(z).z(w).\mathbf{0})
\end{aligned}$$

The processes are composed together inside the SN group.

$$\mathsf{SocialNetwork} = (\nu\ \mathsf{SN})(\mathsf{User} \mid \mathsf{notAFriend} \mid \mathsf{Friend})$$

To achieve this, we define the types

$$\begin{aligned}
\mathsf{T}_{\mathsf{pic}} &= \mathsf{SN}[]^{-*} \\
\mathsf{T}_{\mathsf{paddr}} &= \mathsf{SN}[\mathsf{T}_{\mathsf{pic}}]^{\mathtt{rw}*} \\
\mathsf{T}'_{\mathsf{paddr}} &= \mathsf{SN}[\mathsf{T}_{\mathsf{pic}}]^{-1} \\
\mathsf{T}_{a} &= \mathsf{SN}[\mathsf{T}_{\mathsf{paddr}}]^{\mathtt{rw}*} \\
\mathsf{T}_{\mathsf{tofriend}} &= \mathsf{SN}[\mathsf{T}_{a}]^{\mathtt{rw}0} \\
\mathsf{T}_{\mathsf{public}} &= \mathsf{SN}[\mathsf{T}_{\mathsf{paddr}}]^{\mathtt{rw}0} \\
\mathsf{T}'_{\mathsf{public}} &= \mathsf{SN}[\mathsf{T}'_{\mathsf{paddr}}]^{\mathtt{rw}0}
\end{aligned}$$

which are combined into the following tuples

$$\begin{aligned}
PIC &= (\mathsf{T}_{\mathsf{pic}}, \mathsf{T}_{\mathsf{pic}}) \\
A &= (\mathsf{T}_{a}, \mathsf{T}_{a}) \\
PA &= (\mathsf{T}_{\mathsf{paddr}}, \mathsf{T}_{\mathsf{paddr}}) \\
TF &= (\mathsf{T}'_{\mathsf{tofriend}}, \mathsf{T}_{\mathsf{tofriend}}) \\
PB &= (\mathsf{T}'_{\mathsf{public}}, \mathsf{T}_{\mathsf{public}})
\end{aligned}$$

We can show that:

$$a : A \cdot \mathsf{paddr} : PA, \mathsf{tofriend} : TF \cdot \mathsf{public} : PB \cdot \mathsf{pic} : PIC \vdash \mathsf{SocialNetwork}$$

whereas for $\mathsf{notAFriend}' = \mathsf{public}(z).z(w).\mathbf{0}$ and

$$\mathsf{SocialNetwork}' = (\nu\ \mathsf{SN})(\mathsf{User} \mid \mathsf{notAFriend}' \mid \mathsf{Friend})$$

the following judgment fails.

$$a : A \cdot \mathsf{paddr} : PA, \mathsf{tofriend} : TF \cdot \mathsf{public} : PB \cdot \mathsf{pic} : PIC \vdash \mathsf{SocialNetwork}'$$

## 5 Conclusions

In this paper we have presented a formal framework based on the $\pi$-calculus with groups for studying privacy. Our framework is accompanied by a type system for capturing privacy-related notions: it includes the use of *groups* to enable reasoning about information collection, it builds on *read/write capabilities* to control information processing, and it employs *type linearity* to restrict information dissemination. We illustrate the use of our typing system via simple examples.

In future work we would like to provide a safety criterion for our framework by developing a policy language for defining privacy policies associated to process calculus descriptions and subsequently to refine our type system so that it can check the satisfaction/violation of these policies. Furthermore, we would like to study the relation of our type system to other typing systems in the literature.

## References

1. Cardelli, L., Ghelli, G., Gordon, A.D.: Secrecy and group creation. Inf. Comput. **196**(2), 127–155 (2005)
2. Dardha, O., Giachino, E., Sangiorgi, D.: Session types revisited. In: Proceedings of PPDP'12, pp. 139–150. ACM, New York (2012)
3. Kobayashi, N., Pierce, B.C., Turner, D.N.: Linearity and the pi-calculus. ACM Trans. Program. Lang. Syst. **21**(5), 914–947 (1999)
4. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, parts I and II. Inf. Comput. **100**(1), 1–77 (1992)
5. Pierce, B.C., Sangiorgi, D.: Typing and subtyping for mobile processes. Math. Struct. Comput. Sci. **6**(5), 409–453 (1996)
6. Solove, D.J.: A taxonomy of privacy. Univ. PA Law Rev. **154**(3), 477–560 (2006)
7. Tschantz, M.C., Wing, J.M.: Formal methods for privacy. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 1–15. Springer, Heidelberg (2009)