Type checking privacy policies in the π-calculus

Dimitrios Kouzapas[1] and Anna Philippou[2]

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

Abstract

In this paper we propose a formal framework for studying privacy. Our framework is based on the π-calculus with groups [6] 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. Furthermore, we associate our framework with a privacy policy language and we prove that if a system is well-typed according to a typing that is compatible with a policy then the system respects the policy.

1 Introduction

During recent years, with the advent of network and information technologies, we are witnessing new practices relating to the collection, processing and sharing of personal information. The formation of large databases that aggregate personal information, health care electronic record systems, cell phone companies that collect and use location data about their users, on-line social networks and search engines, are a few examples. While enabling useful services to the public, these practices are arousing great concerns as to how this personal data is used by organizations, and impose new challenges for protecting individuals from violation of their right to privacy as well as for providing solid foundations for understanding privacy as a term. As a result, during the last decade, a great deal of work is concentrating on understanding the types of practices and policies which are appropriate for preserving the privacy rights of individuals in different settings and how to represent and enforce such policies [26, 22].

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 [26], where the authors base their arguments on the taxonomy of privacy rights by Solove [25]. According to [25], 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, the data holder and the environment. In this setting, the data holder possesses information about the data subject and is responsible to protect this information against unauthorized adversaries within the environment.

The goal of this work is to provide a formal framework for reasoning about privacy-related concepts, as discussed above. Such a framework would provide solid foundations for understanding the notion of 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. In our previous work of [18], we used the meta-theoretic framework of the π-calculus to describe and study privacy properties. More precisely, we based our framework on the π-calculus with groups [6] accompanied by a type system for capturing privacy-related notions as discussed in [5]. We showed this type system to be sound via subject reduction. However, this initial

framework was incomplete in the sense that it was not as yet associated with a safety result. Thus, the main goal of this work is to extend our previous work by providing an appropriate safety criterion for our framework. The importance of such a safety theorem is two-fold: On the one hand, it will enable to establish a correspondence between privacy as a legal (and/or philosophical) notion and privacy as a computational notion. On the other hand, it will provide necessary machinery for proving privacy preservation by typing. We present the main ideas of our proposal in the following example.

Motivating Example. Consider a medical database where patient data is stored and can be accessed by a database administrator, a nurse and a doctor. In this context consider the following scenario where the database administrator process *DA* sends a reference c to a patient's data to a doctor process D using a nurse process N 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.

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

In this setting a variety of privacy requirements could be enunciated. To begin with it would be natural to require the following:

> Requirement 1: No external adversary will be able to access patient data.

The use of groups, as proposed in the π-calculus with groups of [6], appears appropriate for dealing with this requirement. The semantics for this calculus disallows the leakage of information (secrets) by proposing 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. This semantics is convenient to achieve privacy properties pertaining to *information collection*: A data holder can use the group type to disallow unauthorized adversaries from collecting information about a data subject.

Thus, in our example, we may compose the above processes under the fresh hospital group Hosp, and employ an appropriate typing of c which specifies that c belongs to group Hosp. In this way it will not be possible for c to be leaked outside the defined context. This is because (1) groups are not values and cannot be communicated and (2) the group Hosp is only known by the three processes (see [6] for the details).

$$(\nu\ \mathsf{Hosp})(((\nu c : \mathsf{Hosp}[])DA) \mid N \mid D)$$

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 the patient's record in contrast to the doctor who is allowed both of these capabilities:

> Requirement 2: A doctor may read and write patient data and a nurse must not read and write patient data.

To address such requirements it is necessary (1) to distinguish between the components of a system (e.g. nurse and doctor) and (2) to reason about the type of access they have on sensitive data. With respect to the first issue, we build on the notion of a group of the calculus and we use the group memberships of processes to distinguish their roles within the system. As far as the second issue is concerned, we turn to the input/output typing system for the π-calculus of Pierce and Sangiorgi, [23]. 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 requirements in question: A type system that controls read and write capabilities[1] can be used to control and observe how the information about a data

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

subject is 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:

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

where names a, b and c are of types T_a, T_b and T_c, respectively, and MedicalData is the basic type of medical data. Channel c can be used for reading and writing medical data. Channel a is used to pass information to the nurse from the database administrator, while channel b is used for the nurse to provide information to the doctor. Both channels can be used for reading and writing. Furthermore, we extend our system with the use of three further groups, namely, DBadmin, Nurse and Doctor and we rewrite our system as:

$$(\nu\ \mathsf{Hosp})(((\nu c : T_c)((\nu\ \mathsf{DBadmin})DA) \mid (\nu\ \mathsf{Nurse})N \mid (\nu\ \mathsf{Doctor})D)$$

In the above system, process DA is nested under groups Hosp and DBadmin. Similarly processes N and D are nested under groups $\{\mathsf{Hosp}, \mathsf{Nurse}\}$ and $\{\mathsf{Hosp}, \mathsf{Doctor}\}$, respectively. We consider these group memberships of processes to characterize the type of the processes: processes with the same group memberships are considered to possess the same distinct role within the system. Of interest to us is the way in which each of these roles accesses sensitive information. In our example, we may use roles/group memberships in two distinct ways. On the one hand may verify that the system is well-typed as all processes use their names in accordance to their type. On the other hand, we may also infer that, for instance, the role $\mathsf{Hosp} \cdot \mathsf{Nurse}$ neither reads nor writes medical data, whereas the role $\mathsf{Hosp} \cdot \mathsf{Doctor}$ reads and writes medical data. As far as our privacy requirement is concerned this is an important piece of information as it allows us to deduce that the system satisfies Requirement 2 above, which in our setting could be expressed as $\mathsf{Hosp} \cdot \mathsf{Nurse} : \emptyset$ and $\mathsf{Hosp} \cdot \mathsf{Doctor} : \mathsf{read}, \mathsf{write}$.

These observations are at the core of our framework. Given a system and a type environment we perform type checking to confirm that the system is well-typed while we infer a type interface for each component of the system. To check that the system complies with some privacy policy we provide a simple language to express privacy policies and we establish a correspondence between the policy language and the type interfaces produced by our type system, the intention being that a typing interface and a policy are compatible if and only if the typing interface exercises a subset of the capabilities permitted by the policy. With this machinery at hand, we state and prove a safety theorem which states that if a system *Sys* type checks against a typing Γ and produces an interface Δ which is compatible with a policy P, then *Sys* respects P.

Our framework also deals with the *information dissemination* category of privacy violations. This is implemented by handling 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 [17]. A linear typing system can be used by the data holder to control the number of times an information can be disseminated and to observe the number of times information is disseminated. In our example, we might require from the nurse the capability of sending the reference of the patient only once, while requiring from the doctor not to share the information with anyone else:

$$\begin{aligned} T_c^0 &= \mathsf{Hosp}[\mathsf{MedicalData}]^{\mathtt{rw0}} \\ T_c^1 &= \mathsf{Hosp}[\mathsf{MedicalData}]^{\mathtt{rw1}} \\ T_a &= \mathsf{Hosp}[\mathsf{Hosp}[T_c^1]]^{\mathtt{rw0}} \\ T_b &= \mathsf{Hosp}[\mathsf{Hosp}[T_c^0]]^{\mathtt{rw0}} \end{aligned}$$

According to types T_a and T_b, channels a and b cannot be sent to other entities. As far as types T_c^1 and T_c^0 are concerned, the first is used to type a name that can be sent at most once, whereas T_c^0 is used to type a name that cannot be sent. Thus, name a can be used to receive a name which can be forwarded once whereas name b can be used to receive a name that cannot be forwarded at all.

Related work. There exists a large body of literature concerned with reasoning about privacy. To begin with, a number of languages have been proposed to express privacy policies [8, 1, 20, 14, 19, 21]. Some of these languages are associated with formal semantics and can be used to verify the consistency of policies or to check whether a system complies with a certain policy. These verifications may be performed *a priori* via static techniques such as model checking [19], on-the-fly using monitoring, e.g. [3, 24], or *a posteriori*, e.g. through audit procedures [9, 2, 11].

More related to our work is the research line on typed-based security in process calculi. Among these works, numerous works have focused on access control which is closely related to privacy. For instance the work on the Dπ calculus has introduced sophisticated type systems for controlling the access to resources advertised at different locations [15, 16]. Furthermore, discretionary access control has been considered in [5] which similarly to our work employs the π-calculus with groups, while role-based access control has been considered in [4, 12, 7]. Finally, we mention that a type system for checking differential privacy for security protocols was developed in [13] for enforcing quantitative privacy properties. However, to the best of our knowledge, there exists no prior work on typing systems for checking compliance against privacy policies as implemented in our framework.

2 The Calculus

Our study of privacy is based on the π-calculus with groups proposed by Cardelli et al. [6] with some modifications. In this section we overview the syntax and reduction semantics of the calculus.

Beginning with the syntax, this is defined at two levels. At the lower level, the process level, we have the standard π-calculus syntax. At the higher level, the system level, we include the group construct, applied both at the level of processes $(\nu\ G)P$, and at the level of systems, $(\nu\ G)S$, the name restriction construct as well as parallel composition for systems. The definition of types is in Section 3.

$$\begin{array}{lcl} P & ::= & x(y{:}T).P \ \mid\ \overline{x}\langle z\rangle.P \ \mid\ (\nu\ a{:}T)P \ \mid\ P_1 \mid P_2 \ \mid\ !P \ \mid\ \mathbf{0} \\ S & ::= & (\nu\ G)P \ \mid\ (\nu\ G)S \ \mid\ (\nu\ a{:}T)S \ \mid\ S_1 \mid S_2 \ \mid\ \mathbf{0} \end{array}$$

Free names $\mathtt{fn}(P)$ are defined in the standard way as for π-calculus processes. We extend this notion to the sets of free groups in a system S and a type T which we denote as $\mathtt{fg}(S)$ 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:

$$\begin{array}{ll} P \mid \mathbf{0} \equiv P & (\nu\ a{:}T)P_1 \mid P_2 \equiv (\nu\ a:T)(P_1 \mid P_2) \text{ if } a \notin \mathtt{fn}(P_2) \\ P_1 \mid P_2 \equiv P_2 \mid P_1 & (\nu\ a{:}T_1)(\nu\ b{:}T_2)P \equiv (\nu\ b{:}T_2)(\nu\ a{:}T_1)P \\ (P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3) & !P \equiv P \mid !P \\ S \mid \mathbf{0} \equiv S & (\nu\ G)(S_1 \mid S_2) \equiv (\nu\ G)S_1 \mid (\nu\ G)S_2 \\ S_1 \mid S_2 \equiv S_2 \mid S_1 & (\nu\ G_1)(\nu\ G_2)S \equiv (\nu\ G_2)(\nu\ G_1)S \\ (S_1 \mid S_2) \mid S_3 \equiv S_1 \mid (S_2 \mid S_3) & (\nu\ G)(\nu\ a{:}T)P \equiv (\nu\ a{:}T)(\nu\ G)P \text{ if } G \notin \mathtt{fg}(T) \end{array}$$

Note that the usual axiom of $(\nu\ G)(S_1 \mid S_2) \equiv (\nu\ G)S_1 \mid S_2$ if $G \notin \mathtt{fg}(S_2)$ is missing from the relation. This is due to our intended semantics of the group concept which is considered to assign capabilities to processes. Thus, nesting of a process P within some group G, as in $(\nu\ G)P$, cannot be lost even if $G \notin \mathtt{fg}(P)$, since the $(\nu\ G)$ construct has the additional meaning of group membership in our calculus and it instills P with privacy-related permissions as we will discuss in the sequel.

We may now present the reduction relation "$\longrightarrow$" which consists of the standard π-calculus reduction relation extended with a new rule for group creation and an adapted rule for parallel composition. The latter is defined at the level of $(\nu\ \widetilde{G})P$ processes where $\widetilde{G}$ is a tuple of groups and if $\widetilde{G} = \langle G_1, \ldots G_n \rangle$ then $(\nu\ \widetilde{G}) = (\nu\ G_1) \ldots (\nu\ G_n)$ whereas if $\widetilde{G} = \langle\rangle$ then $(\nu\ \widetilde{G})P = P$. Note that, in the rules that follow, we write F to range over both processes P and systems S.

$$
\begin{array}{rcll}
(\nu\ \widetilde{G}_1)\overline{a}\langle b\rangle.P_1 \mid (\nu\ \widetilde{G}_2)a(x:T).P_2 & \longrightarrow & & (\nu\ \widetilde{G}_1)P_1 \mid (\nu\ \widetilde{G}_2)P_2\{b/x\} \\
F_1 \longrightarrow F_2 & \text{implies} & & F_1 \mid F_3 \longrightarrow F_2 \mid F_3 \\
F_1 \longrightarrow F_2 & \text{implies} & & (\nu\ G)F_1 \longrightarrow (\nu\ G)F_2 \\
F_1 \longrightarrow F_2 & \text{implies} & & (\nu\ a:T)F_1 \longrightarrow (\nu\ a:T)F_2 \\
F_1 \equiv F_1', F_1' \longrightarrow F_2', F_2' \equiv F_2 & \text{implies} & & F_1 \longrightarrow F_2
\end{array}
$$

3 Types and subtyping

In this section we define a typing system for the calculus which builds upon the typing of [6]. The typing system includes: (i) the notion of groups of [6], (ii) the read/write capabilities of [23] 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 name, its type specifies (1) the group it belongs to (the name should not be disclosed to users who do not belong to this group), (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 λ below). Given the above, a type is constructed via the following BNF's where BT belongs to a set of base types BT.

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

For example, a channel of type $G[T]^{\mathtt{r}\,2}$ is a channel belonging to group G that communicates names of type T, 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 λ, denoted as $\sqsubseteq_\lambda$:

$\sqsubseteq_p$: $\quad \mathtt{rw} \sqsubseteq_p \mathtt{w} \quad \mathtt{rw} \sqsubseteq_p \mathtt{r} \quad \mathtt{rw}, \mathtt{r}, \mathtt{w} \sqsubseteq_p -$

$\sqsubseteq_\lambda$: $\quad * \sqsubseteq_\lambda i$ for all $i \quad i \sqsubseteq_\lambda j$ if $i \geq j$

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 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

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

The first pair in the construction of $\mathscr{F}$ says that the subtyping relation is reflexive on base types. 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: $G_1[G_2[T]^{\mathtt{rw}5}]^{\mathtt{rw}*} \leq G_1[G_2[T]^{\mathtt{w}3}]^{\mathtt{r}0}$, and $G_1[G_2[T]^{-3}]^{\mathtt{rw}*} \leq G_1[G_2[T]^{-3}]^{\mathtt{w}0}$.

4 Type System

Typing Judgements. We now turn to the typing system of our calculus. The environment on which type checking is carried out consists of the component Γ. This declares the names and groups in scope during type checking. We define Γ-environments by

$$\Gamma,\Delta ::= \emptyset \mid \Gamma \cdot x : T \mid \Gamma \cdot G$$

The domain of environment Γ, $\mathtt{dom}(\Gamma)$, is considered to contain all groups and all names recorded in Γ.

We define three typing judgements: $\Gamma \vdash x \triangleright T$, $\Gamma \vdash P \triangleright \Delta$ and $\Gamma \vdash S \triangleright \Theta$. The first typing judgement says that under the typing environment Γ, variable x has type T. The second typing judgement stipulates that process P is well typed under the environment Γ and produces a type environment Δ. In this judgment, Γ records the types of the names of P and Δ records all names in P and how they are used in P. Finally, the third judgment defines that system S is well typed under the environment Γ and produces interface Θ which records the group memberships of all components of S as well as the usage of the channels in each component. A Θ-interface is defined by

$$\Theta ::= \varepsilon \mid \langle G_1 \cdot \ldots \cdot G_n, \Gamma \rangle \cdot \Theta$$

Typing System. We now move on to the rules of our typing system. First, we present some 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. Next, we define the permission addition operator $\oplus$ where $p_1 \oplus p_2 = p_1 p_2$ denotes the merging of the two permissions. We lift these notions to the level of types by defining $T_1 \oplus T_2 = G[T_1' \oplus T_2']^{(p_1 \oplus p_2)(\lambda_1 \oplus \lambda_2)}$, if $T_1 = G[T_1']^{p_1\lambda_1}$ and $T_2 = G[T_2']^{p_2\lambda_2}$.

We may now extend these functions to the level of typing environments. Operator $\uplus$, defined below, composes its arguments by concatenating their declarations with the exception of the common domain where types are added up via $\oplus$:

$$\Gamma_1 \uplus \Gamma_2 \quad = \quad \Gamma_1 \backslash \Gamma_2 \cdot \Gamma_2 \backslash \Gamma_1 \cdot \{x : T_1 \oplus T_2 | x : T_1 \in \Gamma_1, x : T_2 \in \Gamma_2\}$$

Finally, we define type operators $\mathsf{iperm}(T)$ and $\mathsf{operm}(T)$ as:

$$\mathsf{iperm}(T) \quad = \quad \begin{cases} BT & \text{if } T = BT \\ G[T']^{-0} & \text{if } T = G[T']^{p\lambda} \end{cases} \qquad \mathsf{operm}(T) \quad = \quad \begin{cases} BT & \text{if } T = BT \\ G[T']^{-1} & \text{if } T = G[T']^{p\lambda} \end{cases}$$

$$\text{(Name)}\quad \frac{\mathtt{fg}(T)\subseteq\Gamma}{\Gamma\cdot x:T\vdash x\triangleright T} \qquad \text{(Nil)}\quad \Gamma\vdash \mathbf{0}\triangleright\emptyset$$

$$\text{(SubsN)}\quad \frac{\Gamma\vdash x:T' \qquad T'\leq T}{\Gamma\vdash x:T} \qquad \text{(SubsP)}\quad \frac{\Gamma\cdot x:T'\vdash P\triangleright\Delta \qquad T'\leq T}{\Gamma\cdot x:T\vdash P\triangleright\Delta}$$

$$\text{(In)}\quad \frac{\Gamma\cdot y:T\vdash P\triangleright\Delta \qquad \Gamma\vdash x:G_x[T']^{r0} \qquad (\Delta\uplus y:\mathsf{iperm}(T))(y)=T'}{\Gamma\vdash x(y:T).P\triangleright\Delta\uplus y:\mathsf{iperm}(T)\uplus x:G_x[T']^{r0}}$$

$$\text{(Out)}\quad \frac{\Gamma\vdash P\triangleright\Delta \qquad \Gamma\vdash y:T' \qquad \Gamma\vdash x:G_x[T]^{w0} \qquad (\Delta\uplus y:\mathsf{operm}(T'))(y)=T}{\Gamma\vdash \bar{x}\langle y\rangle.P\triangleright\Delta\uplus y:\mathsf{operm}(T')\uplus x:G_x[T]^{w0}}$$

$$\text{(ParP)}\quad \frac{\Gamma_1\vdash P_1\triangleright\Delta_1 \qquad \Gamma_2\vdash P_2\triangleright\Delta_2}{\Gamma_1\uplus\Gamma_2\vdash P_1\mid P_2\triangleright\Delta_1\uplus\Delta_2} \qquad \text{(ParS)}\quad \frac{\Gamma_1\vdash S_1\triangleright\Theta_1 \qquad \Gamma_2\vdash S_2\triangleright\Theta_2}{\Gamma_1\uplus\Gamma_2\vdash S_1\mid S_2\triangleright\Theta_1\cdot\Theta_2}$$

$$\text{(ResNP)}\quad \frac{\Gamma\cdot x:T\vdash P\triangleright\Delta}{\Gamma\vdash(\nu\, x:T)P\triangleright\Delta} \qquad \text{(ResNS)}\quad \frac{\Gamma\cdot x:T\vdash S\triangleright\Theta}{\Gamma\vdash(\nu\, x:T)S\triangleright\Theta}$$

$$\text{(ResGP)}\quad \frac{\Gamma\cdot G\vdash P\triangleright\Delta}{\Gamma\vdash(\nu\, G)P\triangleright\langle G:\Delta\rangle} \qquad \text{(ResGS)}\quad \frac{\Gamma\cdot G\vdash S\triangleright\{\langle\tilde{G}_i,\Delta_i\rangle\}_{i\in I}}{\Gamma\vdash(\nu\, G)S\triangleright\{\langle G,\tilde{G}_i:\Delta_i\rangle\}_{i\in I}}$$

$$\text{(Rep)}\quad \frac{\Gamma\vdash P\triangleright\Delta \qquad \forall x:G[T]^{p\lambda}\in\Delta,\lambda\neq 0\implies\Gamma\vdash x:G[T]^{p*}}{\Gamma\vdash !P\triangleright\Delta^{!}}$$

Figure 1: The Typing System

The typing system is defined in Figure 1. Rule (Name) is used to type names. Note that in name typing we require that all group names of the type are present in component Γ of the typing environment. The inactive process can be typed under any typing environment (axiom (Nil)) to infer an empty type environment. Rule (SubsN) defines a subsumption based on subtyping for channels. Similar to rule (SubsN), rule (SubsP) defines subsumption for process typing judgments.

Rule (In) types the input-prefixed process. We first require that the input subject has at least permission for reading. If, additionally, environment Γ extended with the type of y produces Δ as an interface of P, we conclude that the process $x(y).P$ produces an interface where the type of x is extended with the read-capability of a type T and the least permissions for name y. The rule for the output prefix (Out) checks that the output subject has write permissions. Then, if the continuation process produces a typing Δ, then the output-prefixed process produces an extension of Δ where x possesses a write capability and the linearity of y is increased by 1.

Parallel composition of processes, (ParP), uses the $\uplus$ operator to compose typing interfaces of P_1 and P_2. Parallel composition of systems, rule (ParS), simply concatenates the typing interfaces of S_1 and S_2. For name restriction, (ResNP) specifies that if a process type checks with in an environment Γ extended with the typing $x:T$, then the process with x restricted type checks in environment Γ. (ResNS) is defined similarly. Moving on to group creation, for rule (ResGP) we have that, if P produces a typing Δ, then system $(\nu\ G)P$ produces the interface $\langle G,\Delta\rangle$, whereas for rule (ResGS), we have that if S produces a typing interface $\{\langle\widetilde{G}_i,\Delta_i\rangle\}_{i\in I}$, then process $(\nu\ G)S$ produces interface $\{\langle G\cdot\widetilde{G}_i,\Delta_i\rangle\}_{u\in I}$. Essentially this captures that if a system has a set of components each possessing a set of groups and a set of names, then

enclosing the system within an $(\nu\ G)$ operator results in adding the group to the group memberships of each of the components.

Finally, for the replication operator, axiom (Rep), we have that if P produces a type interface Δ then to type $!P$, we require that all names in Δ with linearity greater than 0 have linearity $*$ in Γ and the interface produced is $\Delta^! = \{x : G[T]^{p0} | x : G[T]^{p0} \in \Delta\} \cup \{x : G[T]^{p*} | x : G[T]^{p\lambda} \in \Delta, \lambda \neq 0\}$.

5 Policies

In this section we define a simple language for enunciating privacy requirements of systems defined in our process calculus. Typically, privacy policy languages express positive and negative norms that are expected to hold in a system. These norms distinguish what *may* happen, in the case of a positive norm, and what may not happen in the system, in the case of a negative norm. Furthermore, they constitute clauses that define what is allowed/disallowed for *data attributes* which are types of sensitive data within a system such as *patient data*, *location* or *telephone number* and, in particular, how the various agents, who are referred to by their *roles* as opposed to their names, may/may not handle this data.

The notions of an attribute and a role are reflected in our framework via the notions of base types and groups, respectively. Thus, our simple policy language is defined in such a way as to specify the allowed and disallowed permissions associated with the various groups for each base type. Specifically, we define the set of possible permissions by

$$\mathsf{Per} = \{\mathsf{read}, \mathsf{write}, \mathsf{forward}\ \lambda, \mathsf{exclude}, \mathsf{nondisclose}\}$$

Note that the first three permissions are used to define positive norms: they can be used to express that data may be read (read), written (write) and forwarded up-to λ times (forward λ). In contrast, the last two norms are negative norms: exclude when associated with a group and a base type it expresses that the group must be excluded from any actions relating to the base type whereas permission nondisclose, when associated with a group and a base type, expresses that the base type cannot be disclosed to any participant who is not a member of the group.

In turn, a policy can be defined via the following BNF's:

$$\mathscr{P} \ ::= \ BT \gg H \ \mid \ \mathscr{P}; \mathscr{P} \qquad\qquad H \ ::= \ G{:}P\,[H_i]_{i\in I}$$

where $P \subseteq \mathsf{Per}$. A policy has the form $BT_1 \gg H_1; \ldots; BT_n \gg H_n$ where BT_i are the base types of the data subject to privacy. The components H_i, which we refer to as *permission hierarchies*, specify the group-permission associations for each base type. A permission hierarchy H has the form $G{:}P\,[H_1, \ldots, H_m]$. The component captures a hierarchy of allowed permissions on the various groups: the outer group G has allowed permissions P and, additionally, the hierarchy permissions of the H_i also hold. Intuitively, this expresses that an entity possessing a group membership in group G has rights P to the data in question and if, additionally it is a member of some group G_i where $H_i = G_i{:}P_i\ [\ldots]$, then it also has the rights P_i, and so on.

We are interested in a subset of policies which we refer to as *legal* policies where a policy $BT_1 \gg H_1; \ldots; BT_n \gg H_n$, is legal if it satisfies that the BT_i are distinct. Additionally, a policy is legal if for any permission hierarchy $G{:}\,[H_1^G, \ldots H_k^G]$ within some H_i then (1) if $\mathsf{nondisclose} \in P$ then nondisclose does not occur in any of the H_i^G's, (2) G does not occur in any of the H_i^G's, and (3) if $\mathsf{exclude} \in P$ then $P = \{\mathsf{exclude}\}$ and $k = 0$. Hereafter, we work with legal policies. As a shorthand, we write $G : P$ for the hierarchy $G{:}P\ [\varepsilon]$ and we abbreviate G for $G : \emptyset$.

As an example, consider a hospital containing doctors, secretaries, patients and janitors where each doctor may belong to a certain department. This may be implemented in a system consisting of groups Hospital, Patient, Doctor, Surgeon, GP, Secretary, Janitor. Further, let us assume the existence of

data of type MedFile, which, should not be disclosed to any participant outside the Hospital group. Furthermore, the data may be read by a patient or a doctor and written by a doctor, whereas a GP may also forward medical files. A janitor should be completely oblivious to the existence of the data while a secretary may neither read nor write the data but may forward it to other system participants. We may capture this requirement via the following policy where $p_n = \{\mathsf{nondisclose}\}$, $p_r = \{\mathsf{read}\}$, $p_{rw} = \{\mathsf{read}, \mathsf{write}\}$, $p_e = \{\mathsf{exclude}\}$ and $p_f = \{\mathsf{forward}\}$:

$$\begin{array}{rcl} \mathscr{P} & = & BT \gg H \\ H & = & \mathsf{Hospital}{:}p_n\,[\mathsf{Patient}{:}\,p_r, \mathsf{Doctor}{:}p_{rw}\,[\mathsf{Surgeon}{:}p_{rw}, \mathsf{GP}{:}p_f], \mathsf{Janitor}{:}p_e] \end{array}$$

This captures that within the group Hospital there exist the groups Patient, Janitor, Doctor, Surgeon and GP, where the group Doctor can be further instantiated as a Surgeon or as a GP. Furthermore, each group has the rights associated with all its group memberships: Any entity that possesses group membership to Hospital must obey the negative norm of non-disclosure, and, thus, it should not disclose patient data, outside of the hospital. Similarly, an entity that is member to groups Hospital and Patient is further endowed with the positive norm of being able to read the data. In the same vein, an entity that possesses membership to the groups Hospital, Doctor and GP has the set of rights $p_n \cup p_{rw} \cup p_f$.

We now proceed to define the notion of *compatibility* between a policy $\mathscr{P}$ and an interface Δ as computed by our type system. To achieve this, it is necessary to provide an association between these two structures. The first step in this direction is achieved by the following functions which extract the permissions exercised with respect to a base type BT by a type interface Θ:

Definition 1.

1. Given a base type BT and a type environment Γ, function perms_Γ is defined as follows:

$$\begin{array}{rcl} \mathsf{perms}_\Gamma(\Gamma, BT) & = & \{\mathsf{exclude} \mid \forall x : T \in \Gamma, \mathtt{fg}(T) \cap BT = \emptyset\} \cup \{\mathsf{read} \mid x : G[BT]^{\mathtt{r}p\,\lambda} \in \Gamma\} \\ & \cup & \{\mathsf{write} \mid x : G[BT]^{\mathtt{w}p\,\lambda} \in \Gamma\} \cup \{\mathsf{forward}\,* \mid x : G[BT]^{p*} \in \Gamma\} \\ & \cup & \{\mathsf{forward}\,\lambda \mid x : G[BT]^{p\lambda_i} \in \Gamma, x : G[BT]^{p*} \notin \Gamma, \lambda = \sum_{i \in I} \lambda_i \neq 0\} \\ & \cup & \{(\mathsf{nondisclose}, G) \mid \forall x : T \in \Gamma, \text{ if } G'[BT]^{p\lambda} \in T \text{ then } G' = G\} \end{array}$$

2. Given a base type BT and a type interface Θ, function perms_Θ is defined as follows:

$$\mathsf{perms}_\Theta(\Theta, BT) = \begin{cases} \emptyset & \text{if } \Theta = \emptyset \\ \mathsf{perms}_\Theta(\Theta', BT) \cdot \langle G_1 \cdot \ldots \cdot G_n : \mathsf{perms}_\Gamma(\Delta) \rangle & \text{if } \Theta = \Theta' \cdot \langle G_1 \cdot \ldots \cdot G_n : \Delta \rangle \end{cases}$$

Function perms_Γ collects the permissions exercised by a type environment Γ. It observes all types of the form $G[BT]^{p\lambda}$, if they exist, and it collects the permissions read, write and forward, according to p and λ. If there exist a number of names with a type $G[BT]^{p\lambda_i}$ where $\lambda_i > 0$ and no name with type $G[BT]^{p*}$ then it records that the data of interest is forwarded $\sum_{i \in I} \lambda_i$ times in total. If there is no name with knowledge of type BT then it concludes that Γ is excluded from any usage of BT. Finally, it collects all pairs $(\mathsf{nondisclose}, G)$ such that $G[BT]^{p\lambda}$ occurs in some type T, where $x : T \in \Gamma$.

This function is then extended to function perms_Θ which is applied iteratively on all elements $\widetilde{G} : \Delta$ of an interface Θ and collecting $\widetilde{G} : \mathsf{perms}_\Gamma(\Delta)$ for each such element, thus recording that a component with group memberships $\widetilde{G}$ exercises permissions $\mathsf{perms}_\Gamma(\Delta)$.

In our next step we process a policy $\mathscr{P}$ as follows:

Definition 2. Consider a policy $\mathscr{P}$ and a base type BT, $\mathsf{flatten}(\mathscr{P},BT)$ is defined as follows:

$$\mathsf{flatten}(\mathscr{P},BT) = \begin{cases} \{\langle G\cdot\widetilde{G}_i, P\oplus P_i\rangle \mid \mathsf{flatten}(H_i) = \langle\widetilde{G}_i,P_i\rangle_{i\in I}\} & \\ \cup\{\langle G,\mathsf{nondisclose}\rangle \mid \mathsf{nondisclose}\in P\} & \text{if } \mathscr{P}=\mathscr{P}';BT\gg G{:}P\,[H_i]_{i\in I} \\ \{\langle G,\mathsf{exclude}\rangle\} & \text{if } \mathscr{P}=\mathscr{P}';BT\gg G{:}\{\mathsf{exclude}\} \end{cases}$$

where $P\oplus P' = \{\mathsf{exclude}\}$, if $\mathsf{exclude}\in P\cup P'$, and $P\oplus P' = P\cup P' - \{\mathsf{nondisclose}\}$, otherwise.

Note that $\mathsf{flatten}(\mathscr{P},BT)$, flattens the hierarchy of H, where $BT\gg H\in\mathscr{P}$, by collecting together the complete list of group memberships that leads to a certain set of permissions. The set of permissions is computed by merging together permissions as one moves down into the hierarchy (in this sense a group inherits all permissions endowed to groups at higher levels of the hierarchy) with the exception of permission exclude: If a group is excluded from access to a base type BT then it automatically loses any permissions that may be available to groups at higher levels.

We may now define the notion of *compatibility* between a type interface Θ and a policy $\mathscr{P}$ as follows:

Definition 3. Given a policy $\mathscr{P}$ and an interface Θ, we define $\mathscr{P}\asymp\Theta$, pronounced $\mathscr{P}$ is compatible to Θ, if for each BT the following hold:

1. If $\langle\widetilde{G}:P\rangle\in\mathsf{perms}_\Theta(\Theta,BT)$ then there exists $\langle\widetilde{G}:Q\rangle\in\mathsf{flatten}(\mathscr{P},BT)$ and $P\subseteq Q$.
2. If $\langle\widetilde{G}:\{\mathsf{exclude}\}\rangle\in\mathsf{flatten}(\mathscr{P},BT)$ then $\langle\widetilde{G}:\{\mathsf{exclude}\}\rangle\in\mathsf{perms}_\Theta(\Theta,BT)$.
3. If $\langle\widetilde{G}:(\mathsf{nondisclose},G)\rangle\in\mathsf{perms}_\Theta(\Theta,BT)$ then $\langle G,\mathsf{nondisclose}\rangle\in\mathsf{flatten}(\mathscr{P},BT)$.

According to the definition a policy $\mathscr{P}$and a type interface Θ are compatible if: (1) any permissions P exercised by a set of groups $\widetilde{G}$ in Θ is allowed by the policy for the specific set of groups; (2) if the policy excludes a group G from the usage of BT-data for some base type BT, then this usage is also excluded from interface Θ; (3) if according to the type interface, the usage of a base type BT is confined within some group G, then all lists of group memberships in $\mathscr{P}$ containing G and whose permissions are not exclude include the nondisclose permission.

6 Soundness and Safety

In this section we prove the properties of the typing system and that typed process are safe with respect to a policy.

The first lemma is standard weakening for the typing environment Γ.

Lemma 1 (Weakening)**.**

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

Proof. The proof is a standard induction on the syntax of processes. □

The converse of weakening for the typing environment Γ is strengthening.

Lemma 2 (Strengthening)**.**

1. If $\Gamma\cdot y:T'\vdash x\triangleright T$, $y\neq x$, then $\Gamma\vdash x\triangleright T$.

2. If $\Gamma \cdot y : T \vdash S \triangleright \Theta$ and $y \notin \mathtt{fn}(P)$ then $\Gamma \vdash S \triangleright \Theta$.

Proof. The proof is a standard induction on the syntax of processes. □

Typing is preserved under substitution. The substitution lemma is important to prove subject reduction.

Lemma 3 (Substitution)**.** If $\Gamma \cdot x : T \vdash P \triangleright \Delta \cdot x : T'$ and $\Gamma \vdash y \triangleright T$ then $\Gamma \triangleright P\{y/x\} \triangleright \Delta \uplus y : T'$

The next definition defines an operator that captures the changes on the interface environment when a process reduces. The interface change is clarified through the subject reduction theorem below.

Definition 4 ($\Theta \leq \Theta'$)**.**

1. $\Delta \leq \Delta'$ if
   - $\forall x : T' \in \Delta'$ such that $x : T \in \Delta$ then $T \leq T'$.
   - $\forall x : T \in \Delta'$ such that $x \notin \mathtt{dom}(\Delta)$ then $\exists y : T \in \Delta$
2. $\Theta \leq \Theta'$ if $\tilde{G} : \Delta \in \Theta$ then $\tilde{G} : \Delta' \in \Theta'$ with $\Delta \leq \Delta'$

Specifically, when a process reduces we expect a name to maintain or lose its interface capabilities, that are expressed through the typing of the name. Also in the case of replicated bound names we expect a possible new name to appear in the interface environment.

An important property of the above definition is that the $\leq$ operator preserves policy compatibility. Intuitively this result will be used to show that processes that reduce maintain their policy compatibility.

Lemma 4. *If Θ is compatible with policy $\mathscr{P}$ and $\Theta \leq \Theta'$ then Θ' is compatible with policy $\mathscr{P}$.*

Proof. The lemma is proved using the induction principle. We show that if $\Theta' \cdot \tilde{G} : \Delta \cdot x : T'$ is compatible with policy $\mathscr{P}$ and $T' \leq T$ then $\Theta \cdot \tilde{G} : \Delta \cdot x : T$ is compatible with policy $\mathscr{P}$.

To prove the above result we do a case analysis on the subtyping relation $T' \leq T$.

□

The typing system enjoys Subject Congruence. Note that systems are congruent and alpha-renamed so that all names in the system are distinct.

Lemma 5 (Subject Congruence)**.** If $\Gamma \vdash S_1 \triangleright \Theta$ and $S_1 \equiv S_2$ then $\Gamma \triangleright S_2 \triangleright \Theta'$ and $\Theta \leq \Theta'$.

We are now ready to state the Subject Reduction theorem. Subject reduction, although standard, it is important because it shows the property that the typing system is closed under the reduction relation and furthermore it captures the changes on the interface environment Θ.

Theorem 1 (Subject Reduction)**.** Let $\Gamma \vdash S \triangleright \Theta$ and $S \longrightarrow S'$ then $\Gamma \vdash S' \triangleright \Theta'$ and $\Theta \leq \Theta'$.

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

$S = (\nu\ G)(\overline{a}\langle b\rangle.P_1 \mid a(x).P_2) \longrightarrow (\nu\ G)(P_1 \mid P_2\{b/x\})$ and $\Gamma \vdash S \triangleright \Theta$ with $\Theta = \langle G : \Delta\rangle$

The typing derivation ends with the rule (ResGS) and in the derivation tree there is the node for the typing of $\overline{a}\langle b\rangle.P_1 \mid a(x).P_2$ which is

$$\Gamma \vdash \overline{a}\langle b\rangle.P_1 \mid a(x).P_2 \triangleright \Delta$$

From the typing system we get that

$$\Gamma = \Gamma_1 \uplus \Gamma_2 \tag{1}$$
$$\Gamma_1 \vdash \overline{a}\langle b\rangle.P_1 \triangleright \Delta_1 \uplus a : G_a[T]^{\mathtt{w}0} \uplus b : T \tag{2}$$
$$\Gamma_2 \vdash a(x).P_2 \triangleright \Delta_2 \uplus a : G_a[T]^{\mathtt{r}0} \uplus x : T \tag{3}$$

From the typing system we get that $\Gamma_1 \vdash P_1 \triangleright \Delta_1$ for (2) and $\Gamma_2 \cdot x : T \cdot P_2 \triangleright \Delta_2$ for (3). We use the substitution lemma (Lemma 3) to get that $\Gamma_2 \vdash P_2\{b/x\} \triangleright \Delta_2 \uplus b : T$.

If we use the (ParP) rule we get that $\Gamma \vdash P_1 \mid P_2\{b/x\} \triangleright \Delta_1 \uplus \Delta_2 \uplus b : T$. The typing is completed with rule (ResGS) to get that $\Gamma \vdash (\nu\ G)(P_1 \mid P_2\{b/x\}) \triangleright \langle G : \Delta_1 \uplus \Delta_2 \uplus b : T\rangle$. Furthermore we conclude with $\Delta \leq \Delta_1 \uplus \Delta_2 \uplus b : T$ because $\Delta = \Delta_1 \uplus a : G_a[T]^{\mathtt{w}0} \uplus b : T \uplus \Delta_2 \uplus a : G_a[T]^{\mathtt{r}0} \uplus x : T$.
Inductive Step:
Case: Parallel Composition. Let $(\nu\ G)(P_1 \mid P_2) \longrightarrow (\nu\ G)(P_1' \mid P_2)$ with $\Gamma \vdash (\nu\ G)(P_1 \mid P_2) \triangleright \Theta$. From the induction hypothesis and the typing derivation of P_1 we know that $\Gamma_1 \vdash P_1 \triangleright \Delta_1$ and $\Gamma_1 \vdash P_1' \triangleright \Delta_1'$ with $\Delta_1 \leq \Delta_1'$. From these two results and the parallel composition typing we can conclude that $\Gamma_1 \uplus \Gamma_2 \triangleright P_1 \mid P_2 \triangleright \Delta_1 \uplus \Delta_2$ and $\Gamma_1 \uplus \Gamma_2 \vdash P_1' \mid P_2 \triangleright \Delta_1' \uplus \Delta_2$. We note that $\Delta_1 \uplus \Delta_2 \leq \Delta_1' \uplus \Delta_2$ We use rule (ResGS) to get $\Gamma_1 \uplus \Gamma_2 \vdash (\nu\ G)(P_1' \mid P_2) \triangleright \langle G : \Delta_1' \uplus \Delta_2\rangle$ and conclude with $\langle G : \Delta_1 \uplus \Delta_2\rangle \leq \langle G : \Delta_1' \uplus \Delta_2\rangle$.
The rest of the inductive step cases are similar to the parallel composition reduction rule. □

We define an auxiliary function on processes to count the send prefixes of names that have an object with base BT.

Definition 5 (Count Send Base Types)**.**

$$\mathsf{countBT}(\mathbf{0}, BT) = 0 \qquad \mathsf{countBT}(\overline{x}\langle y\rangle.P, BT) = 1 + \mathsf{countBT}(P, BT) \text{ if } y \text{ is of type } BT$$
$$\mathsf{countBT}(x(y).P, BT) = \mathsf{countBT}(P, BT) \qquad \mathsf{countBT}(P_1 \mid P_2, BT) = \mathsf{countBT}(P_1, BT) + \mathsf{countBT}(P_2, BT)$$
$$\mathsf{countBT}((\nu\ x)P, BT) = \mathsf{countBT}(P, BT) \qquad \mathsf{countBT}(!P, BT) = \begin{cases} 0 & \text{if } \mathsf{countBT}(P, BT) = 0 \\ * & \text{if } \mathsf{countBT}(P, BT) > 0 \end{cases}$$

The error process clarifies the compatibility relation between the policies and processes. We expect a process not to do something more than a policy is dictating. For example if a policy does not dictate a read permission on a certain level of the hierarchy then a process cannot do a send on private data on the same level of the group hierarchy.

Definition 6 (Error Process)**.** Let system $S \equiv (\nu\ G_1)((\nu\ G_2)(\ldots((\nu\ G_n)P \mid S_n)\ldots) \mid S_1)$ policy $\mathscr{P}$ and x a channel to data of type BT. *System S is an error with respect to* $\mathscr{P}$ if:

1. $\langle G_1 \ldots G_n, p\rangle \in \mathsf{flatten}(\mathscr{P}, BT), \mathsf{read} \notin p$ and $\exists x$ such that $P = x(y).P'$.
2. $\langle G_1 \ldots G_n, p\rangle \in \mathsf{flatten}(\mathscr{P}, BT), \mathsf{write} \notin p$ and $\exists x$ such that $P = \overline{x}\langle y\rangle.P'$.
3. $\langle G_1 \ldots G_n, p\rangle \in \mathsf{flatten}(\mathscr{P}, BT), \mathsf{forward} \notin p$ and $\exists x$ such that $P = \overline{x}\langle y\rangle.P'$
4. $\langle G_1 \ldots G_n, p \cup \{\mathsf{forward}\ n\}\rangle \in \mathsf{flatten}(\mathscr{P}, BT), n \neq *$ and $\mathsf{countBT}(P, BT) > n$
5. $\langle G_1 \ldots G_n, p \cup \{\mathsf{nondisclose}\}\rangle, \langle G_k : \mathsf{nondisclose}\rangle \in \mathsf{flatten}(\mathscr{P}, BT)$, $1 \leq k \leq n$, $\exists x$ with group type G such that $G \notin \{G_k \ldots G_n\}$ and $P = \overline{x}\langle y\rangle.P'$

The first three error processes expect that a process with no read, write or forward (i.e. linearity typing) permissions on a certain level of hierarchy respectively, should not have a subject receive, subject send or object send prefix on private data respectively, on the same level of hierarchy. The fourth error process uses the $\mathsf{countBT}(P, BT)$ function to count the send prefixes on private date and compare them with the forward n policy permission on the same level of the hierarchy. Finally the last error process

describes that a process restricted by a nondisclose policy should not emit private data outside the non-disclosed level of the hierarchy.

It is expected that an error process with respect to a policy $\mathscr{P}$ is not compatible to $\mathscr{P}$.

Lemma 6. Let process S an error process with respect to policy $\mathscr{P}$. If $\Gamma \vdash S \triangleright \Theta$ then interface Θ is not compatible with policy $\mathscr{P}$

Proof. From the definition of the error process we have that $S \equiv (\nu\, G_1)((\nu\, G_2)(\dots((\nu\, G_n)P \mid S_n)\dots) \mid S_1)$.

We do an analysis on the definition cases of the error process S.

Case: 1

$\langle G_1 \dots G_n, p\rangle \in \mathsf{flatten}(\mathscr{P}, BT), \mathsf{read} \notin p$ and $\exists x$ a channel to data of type BT such that $P = x(y).P'$.

If we type S the typing derivation node (rule (In)) for P would be:

$$\frac{\Gamma \cdot y : BT \vdash P' \triangleright \Delta \qquad \Gamma \vdash x : G[BT]^{r0} \qquad (\Delta \uplus y : BT) = BT}{\Gamma \vdash x(y).\triangleright \Delta \cdot y : BT \uplus G[BT]^{\mathtt{r0}}}$$

The typing of S is completed with the application of rule (ResGP) followed by the application of a series of rules (ResGS):

$$\Gamma \vdash S \triangleright \Theta' \cdot \langle G_1 \dots G_n : \Delta \cdot y : BT \uplus G[BT]^{\mathtt{r0}}\rangle$$

From the fact that $\langle G_1 \dots G_n : p\rangle \in \mathsf{flatten}(\mathscr{P}, BT)$ with $\mathsf{read} \notin p$ we get that $\langle G_1 \dots G_n : p' \cup \{\mathsf{read}\}\rangle \in \mathsf{perms}_\Theta(\Theta, BT) \not\subseteq \mathsf{flatten}(\mathscr{P}, BT)$. This concludes that Θ is not compatible with $\mathscr{P}$.

Case: 2, 3

Similar arguments with case 1.

Case: 4

$\langle G_1 \dots G_n, p \cup \{\mathsf{forward}\ n\}\rangle \in \mathsf{flatten}(\mathscr{P}, BT), n \neq *$ and $\mathsf{countBT}(P, BT) > n$.

If we type S the typing for P would be

$$\Gamma \vdash P \triangleright \Delta$$

and $\forall x : G[T]^{p\lambda_i} \in \Delta, \lambda_i \neq *$ and $\sum_{i\in I} \lambda_i > n$ because the sum of all the prefixes that carry a base type are $\mathsf{countBT}(P, BT) > n$.

The typing of S is completed with the application of rule (ResGP) followed by the application of a series of rules (ResGS):

$$\Gamma \vdash S \triangleright \Theta' \cdot \langle G_1 \dots G_n : \Delta\rangle$$

If we flatten Θ we get that $G_1 \dots G_n, p \cup \{\mathsf{forward}\ m\} \in \mathsf{flatten}\Theta, BT$ with $m > n$. We conclude that Θ is not compatible with $\mathscr{P}$.

Case: 5

$\langle G_1 \dots G_n, p \cup \{\mathsf{nondisclose}\}\rangle \in \mathsf{flatten}(\mathscr{P}, BT)$ and $\exists x$ a channel to data of type BT with group type G such that $G \notin \{G_1 \dots G_n\}$ and $P = \bar{x}\langle y\rangle.P'$

If we type S the typing derivation node (rule (Out)) for P would be:

$$\frac{\Gamma \vdash P' \triangleright \Delta \qquad \Gamma \vdash y \triangleright BT \Gamma \vdash x \triangleright G[BT]^{\mathtt{w0}} \qquad (\Delta \uplus y : BT)(y) = BT}{\Gamma \vdash \bar{x}\langle y\rangle.P' \Delta \uplus y : BT \uplus x : G[BT]^{\mathtt{w0}}}$$

The typing of S is completed with the application of rule (ResGP) followed by the application of a series of rules (ResGS):

$$\Gamma \vdash S \triangleright \Theta' \cdot \langle G_1 \dots G_n : \Delta \cdot y : BT \uplus G[BT]^{\mathtt{w0}}\rangle$$

From the last result have that $\{\mathsf{nondisclose}, G\} \notin \mathsf{perms}_\Theta(\Theta, BT)$ which in turn implies that Θ is not compatible with $\mathscr{P}$. □

The next theorem shows that typed process preserve policy compatibility on reduction semantics.

Theorem 2 (Safety)**.** If $\Gamma \vdash S \triangleright \Theta$ and interface Θ is compatible with policy $\mathscr{P}$ and $S \longrightarrow^* S'$ then S' is not an error with respect to policy $\mathscr{P}$.

Proof. If $\Gamma \vdash S \triangleright \Theta$ and $S \longrightarrow^* S'$ then by Theorem 1 we have that $\Gamma \vdash S' \triangleright \Theta'$. By Lemma 4 we have that Θ' is compatible with policy $\mathscr{P}$.

Let S' be an error process with respect to policy $\mathscr{P}$. By Lemma 6 we have that interface Θ' is not compatible with policy $\mathscr{P}$, results that leads to a contradiction. Thus S' is not an error process. □

7 Example

Electronic Traffic Pricing (ETP) is an electronic toll collection scheme in which the fee to be paid by drivers depends on the road usage of their vehicles where factors such as the type of roads used and the times of the usage will determine the toll. To achieve this, for each individual vehicle detailed time and location information must be collected and processed and the correct amounts due can be calculated with the help of a digital tariff and a road map. A number of possible implementation schemes may be considered for this system [10]. In the centralized approach, all location information is communicated to the pricing authority which periodically computes the fee to be paid based on the received information. In the decentralized approach the fee is computed locally on the car via the use of a third trusted entity such as a smart card. Finally, in the commitment-scheme approach the fee is again computed locally but this time without the use of a smart card while the pricing authority is provided with hashes of all locations as evidence that the calculation was computed correctly. In the following subsections we consider the first two approaches and their associated privacy characteristics.

7.1 The centralized approach

This approach makes use of on board equipment (OBE) which computes regularly the geographical position of the car and forwards it to the Pricing Authority (PA) which in turns computes the fee to be paid based on the received information. To avoid drivers tampering with their OBE and communicating false information, the authorities may perform checks on the spot to confirm that the OBE is sending the correct information.

In the $G\pi$-calculus we may model the system with the aid of five groups: ETP corresponds to the entirety of the ETP system, Car refers to the car components which are divided to the OBE and the GPS subgroups, and PA which refers to the pricing authority. As far as types are concerned, we assume the existence of two base types: Loc referring to the attribute of locations and Fee referring to the attribute of fees, and we write $T_l = \mathsf{ETP}[\mathsf{Loc}]^{\mathtt{rw}*}$, $T_r = \mathsf{Car}[T_l]^{\mathtt{rw}0}$, $T_{pa} = \mathsf{ETP}[T_l]^{\mathtt{rw}0}$, $T_x = \mathsf{ETP}[T_l]^{\mathtt{rw}1}$ and $T_{sc} = \mathsf{ETP}[T_x]^{\mathtt{rw}0}$.

$$
\begin{array}{rcl}
O & = & !read(loc : T_l).\overline{topa}\langle loc\rangle.\mathbf{0} \\
 & & \mid !spotcheck(s : T_x).read(ls : T_l).\bar{s}\langle ls\rangle.\mathbf{0} \\
L & = & !(\nu\ newl : T_l)\overline{read}\langle newl\rangle.\mathbf{0} \\
A & = & !topa(z : T_l).z(l : \mathsf{Loc}).\mathbf{0} \\
 & & \mid !\overline{send}\langle fee\rangle.\mathbf{0} \\
 & & \mid !(\nu\ x : T_x)\overline{spotcheck}\langle x\rangle.x(y : T_l).y(l_s : \mathsf{Loc}).\mathbf{0} \\
\mathsf{System} & = & (\nu\ \mathsf{ETP})(\nu\ spotcheck : T_{sc})(\nu\ topa : T_{pa}) \\
 & & [\ (\nu\ \mathsf{PA})A \ \mid \ (\nu\ \mathsf{Car})((\nu\ read : T_r)((\nu\ \mathsf{OBE})O \mid (\nu\ \mathsf{GPS})L))\ ]
\end{array}
$$

In the above model we focus on the communication of information among the system components. To begin with we have the component of the OBE, O, belonging to group OBE, and the component responsible for computing the current location, L, belonging to group GPS. These two components are nested within the Car group and share the private name *read* on which it is possible for L to pass on information of the current location to O. This information is a name via which the current location may be read. The OBE O may spontaneously read on name *read* or it may enquire the current location if required for the purposes of a spot check. Such a check is initiated by the pricing authority A who may engage in three different activities: Firstly, it may receive a name z from the OBE via channel *pa*. This channel may subsequently be used for reading the location of the car (action $z(l)$). Secondly, it may periodically compute the fee to be paid and communicate the link ($fee : \mathsf{ETP}[\mathsf{Fee}]^{\mathtt{rw}*}$) via name $send : \mathsf{ETP}[\mathsf{ETP}[\mathsf{Fee}]^{\mathtt{rw}*}]^{\mathtt{rw0}}$. Thirdly, it may initiate a spot check, during which it creates and send the OBE a new channel via which the OBE is expected to return the current location for a verification check.

By applying the rules of our type system we may show that $\Gamma \vdash \mathsf{System} \triangleright \Theta$, where $\Gamma = \{fee : \mathsf{ETP}[\mathsf{Fee}]^{\mathtt{rw}*}, send : \mathsf{ETP}[\mathsf{ETP}[\mathsf{Fee}]^{\mathtt{rw}*}]^{\mathtt{rw0}}\}$ and where

$$
\begin{array}{rcl}
\Theta & = & \langle \mathsf{ETP}\cdot\mathsf{PA} : \{l : \mathsf{Loc}, z : \mathsf{ETP}[\mathsf{Loc}]^{\mathtt{r0}}, topa : \mathsf{ETP}[\mathsf{ETP}[\mathsf{Loc}]^{\mathtt{r0}}]^{\mathtt{r0}}, \\
 & & \qquad fee : \mathsf{ETP}[\mathsf{Fee}]^{-*}, send : \mathsf{ETP}[\mathsf{ETP}[\mathsf{Fee}]^{-*}]^{\mathtt{w0}} \\
 & & \qquad l_s : \mathsf{Loc}, y : \mathsf{ETP}[\mathsf{Loc}]^{\mathtt{r0}}, x : \mathsf{ETP}[\mathsf{ETP}[\mathsf{Loc}]^{\mathtt{r0}}]^{\mathtt{r}*}, \\
 & & \qquad spotcheck : \mathsf{ETP}[\mathsf{ETP}[\mathsf{ETP}[\mathsf{Loc}]^{\mathtt{r0}}]^{\mathtt{r1}}]^{\mathtt{w0}}\}, \\
 & & \mathsf{ETP}\cdot\mathsf{Car}\cdot\mathsf{OBE} : \{loc : \mathsf{ETP}[\mathsf{Loc}]^{-*}, topa : \mathsf{ETP}[\mathsf{ETP}[\mathsf{Loc}]^{-1}]^{\mathtt{w0}}, \\
 & & \qquad l_s : \mathsf{ETP}[\mathsf{Loc}]^{-*}, s : \mathsf{ETP}[\mathsf{ETP}[\mathsf{Loc}]^{-1}]^{\mathtt{w0}}, \\
 & & \qquad spotcheck : \mathsf{ETP}[\mathsf{ETP}[\mathsf{ETP}[\mathsf{Loc}]^{-1}]^{\mathtt{w0}}]^{\mathtt{r0}} \\
 & & \qquad read : \mathsf{Car}[\mathsf{ETP}[\mathsf{Loc}]^{-1}]^{\mathtt{r0}}\}, \\
 & & \mathsf{ETP}\cdot\mathsf{Car}\cdot\mathsf{GPS} : \{newl : \mathsf{ETP}[\mathsf{Loc}]^{-*}, read : \mathsf{Car}[\mathsf{ETP}[\mathsf{Loc}]^{-1}]^{\mathtt{w0}}\}\rangle
\end{array}
$$

If we apply function perms_Θ on Θ for base type Loc we obtain the following:

$$\mathsf{perms}_\Theta(\Theta, \mathsf{Loc}) = \{\mathsf{ETP}\cdot\mathsf{PA} : \{\mathsf{read}\}, \mathsf{ETP}\cdot\mathsf{Car}\cdot\mathsf{OBE} : \{\mathsf{forward}*\}, \mathsf{ETP}\cdot\mathsf{Car}\cdot\mathsf{GPS} : \{\mathsf{forward}*\}\}$$

A possible privacy policy for this system might be one that states that the location attribute may be freely forwarded by the OBE:

$$
\begin{array}{l}
\mathsf{Loc} \gg \mathsf{ETP} : \mathsf{nondisclose}\ [\\
\qquad\qquad \mathsf{Car} : [\\
\qquad\qquad\qquad \mathsf{OBE} : \{\mathsf{forward}*\} \\
\qquad\qquad\qquad \mathsf{GPS} : \{\mathsf{forward}*\}], \\
\qquad\qquad \mathsf{PA} : \{\mathsf{read}\}]
\end{array}
$$

We may verify that Θ is compatible with the above policy thus the system satisfies the policy.

This architecture is simple, but also very weak in protecting the privacy of individuals: the fact that the PA gets detailed travel information about every vehicle constitutes a privacy and security threat. The central database where this information is stored will be an attractive target for individuals or organisations with unfriendly intentions, like terrorists or blackmailers. In our system this privacy threat can be pinpointed to attribute Loc and the fact that references to locations may be communicated to the PA for an unlimited number of times via the bound names *loc* and l_s. An alternative implementation that limits the transmission of such names is presented in the second implementation proposal presented below.

7.2 The decentralized approach

To avoid the disclosure of the complete travel logs of a system this solution employs a third trusted entity (e.g. smart card) to make computations of the fee locally on the car and send this price to the authority which in turn may make spot checks to obtain evidence on the correctness of the calculation.

The policy here would require that locations can be communicated for at most a small fixed amount of times and that the OBE may read the fee computed by the smart card but not change its value. Precisely, the new privacy policy might be:

$$
\begin{array}{l}
\mathsf{Loc} \gg \mathsf{ETP} : \mathsf{nondisclose}\ [\\
\quad \mathsf{Car} : [\\
\quad\quad \mathsf{OBE} : \{\mathsf{forward}\ 2\} \\
\quad\quad \mathsf{GPS} : \{\mathsf{forward}\ *\} \\
\quad\quad \mathsf{SC} : \{\mathsf{read}\}], \\
\quad \mathsf{PA} : \{\mathsf{read}\} \\
\quad]
\end{array}
\qquad
\begin{array}{l}
\mathsf{Fee} \gg \mathsf{ETP} : \mathsf{nondisclose}\ [\\
\quad \mathsf{Car} : [\\
\quad\quad \mathsf{OBE} : \{\mathsf{read}\} \\
\quad\quad \mathsf{GPS} : \{\} \\
\quad\quad \mathsf{SC} : \{\mathsf{write}, \mathsf{forward}*\}], \\
\quad \mathsf{PA} : \{\mathsf{read}\} \\
\quad]
\end{array}
$$

The new system as described above may be modeled as follows, where we have a new group SC and a new component S, a smart card, belonging to this group:

$$
\begin{array}{rcl}
S & = & !read(loc : T_l).loc(l : \mathsf{Loc}).(\nu\ newval : Fee)\overline{fee}\langle newval\rangle.\overline{send}\langle fee\rangle.\mathbf{0} \\
O & = & spotcheck(s_1 : T_x).read(ls_1 : T_l).\overline{s_1}\langle ls_1\rangle.spotcheck(s_2 : T_x).read(ls_2 : T_l).\overline{s_2}\langle ls_2\rangle.\mathbf{0} \\
L & = & !(\nu\ newl : T_l)\overline{read}\langle newl\rangle.\mathbf{0} \\
A & = & !(\nu\ x : T_x)\overline{spotcheck}\langle x\rangle.x(y : T_l).y(l_s : \mathsf{Loc}).\mathbf{0} \\
 & & \mid send(fee).fee(v : \mathsf{Fee}).\mathbf{0} \\
\mathsf{System} & = & (\nu\ \mathsf{ETP})(\nu\ spotcheck : T_{sc})(\nu\ topa : T_{pa}) \\
 & & [\ (\nu\ \mathsf{PA})A \ \mid\ (\nu\ \mathsf{Car})((\nu\ read : T_r)((\nu\ \mathsf{OBE})O \mid (\nu\ \mathsf{GPS})L) \mid (\nu\ \mathsf{SC})S)\ ]
\end{array}
$$

We may verify that $\Gamma \vdash \mathsf{System} \triangleright \Theta$, where $\Gamma = \{fee : \mathsf{ETP}[\mathsf{Fee}]^{\mathtt{rw}*}, send : \mathsf{ETP}[\mathsf{ETP}[\mathsf{Fee}]^{\mathtt{rw}*}]^{\mathtt{rw}0}\}$ and Θ is such that

$$
\begin{array}{rcl}
\mathsf{perms}_\Theta(\Theta, \mathsf{Loc}) & = & \{\mathsf{ETP}\cdot\mathsf{PA} : \{\mathsf{read}\}, \mathsf{ETP}\cdot\mathsf{Car}\cdot\mathsf{OBE} : \{\mathsf{forward}\,2\}, \\
 & & \mathsf{ETP}\cdot\mathsf{Car}\cdot\mathsf{GPS} : \{\mathsf{forward}\,*\}, \mathsf{ETP}\cdot\mathsf{Car}\cdot\mathsf{SC} : \{\mathsf{read}\}\} \\
\mathsf{perms}_\Theta(\Theta, \mathsf{Fee}) & = & \{\mathsf{ETP}\cdot\mathsf{PA} : \{\mathsf{read}\}, \mathsf{ETP}\cdot\mathsf{Car}\cdot\mathsf{OBE} : \{\}, \\
 & & \mathsf{ETP}\cdot\mathsf{Car}\cdot\mathsf{GPS} : \{\}, \mathsf{ETP}\cdot\mathsf{Car}\cdot\mathsf{SC} : \{\mathsf{write}, \mathsf{forward}\,*\}\}
\end{array}
$$

Based on this, it is we may see that the type interface Θ is compatible with the above policy thus the system satisfied the policy.

8 Conclusions

In this paper we have presented a formal framework based on the π-calculus with groups for studying privacy. Our framework is accompanied by a type system for capturing privacy-related notions and a privacy language for expressing privacy policies. We have proved a subject reduction and a safety theorem for our framework where the latter states that if a system *Sys* type checks against a typing T and T is compatible with a policy P, then *Sys* complies to P. Consequently, this result allows us to decide whether a system satisfies a privacy policy by determining whether the system type checks against a typing that is compatible with the policy.

In future work, we would like to extend our framework in order to encompass reasoning about additional privacy violation instances (e.g. identification and aggregation) as well as the possibility of implementing the theory developed into a tool. We are also interested in exploring a more dynamic setting where the roles held by an agent may evolve over time.

References

[1] M. Backes, B. Pfitzmann, and M. Schunter. A toolkit for managing enterprise privacy policies. In *Proceedings of ESORICS'03*, LNCS 2808, pages 162–180, 2003.

[2] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Proceedings of S&P'06*, pages 184–198, 2006.

[3] D. A. Basin, F. Klaedtke, and S. Müller. Policy monitoring in first-order temporal logic. In *Computer Aided Verification, 22nd International Conference, CAV 2010*, pages 1–18, 2010.

[4] C. Braghin, D. Gorla, and V. Sassone. Role-based access control for a distributed calculus. *Journal of Computer Security*, 14(2):113–155, 2006.

[5] M. Bugliesi, D. Colazzo, S. Crafa, and D. Macedonio. A type system for discretionary access control. *Mathematical Structures in Computer Science*, 19(4):839–875, 2009.

[6] L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. *Information and Computation*, 196(2):127–155, 2005.

[7] A. B. Compagnoni, E. L. Gunter, and P. Bidinger. Role-based access control for boxed ambients. *Theoretical Computer Science*, 398(1-3):203–216, 2008.

[8] L. F. Cranor. *Web privacy with P3P - the platform for privacy preferences*. O'Reilly, 2002.

[9] A. Datta, J. Blocki, N. Christin, H. DeYoung, D. Garg, L. Jia, D. K. Kaynar, and A. Sinha. Understanding and protecting privacy: Formal semantics and principled audit mechanisms. In *Proceedings of ICISS'11*, pages 1–27, 2011.

[10] W. de Jonge and B. Jacobs. Privacy-friendly electronic traffic pricing via commits. In *Proceedings of FAST'08*, LNCS 5491, pages 143–161, 2009.

[11] H. DeYoung, D. Garg, L. Jia, D. K. Kaynar, and A. Datta. Experiences in the logical specification of the hipaa and glba privacy laws. In *Proceedings of WPES'10*, pages 73–82, 2010.

[12] M. Dezani-Ciancaglini, S. Ghilezan, S. Jaksic, and J. Pantovic. Types for role-based access control of dynamic web data. In *Proceedings of WFLP'10*, LNCS 6559, pages 1–29, 2010.

[13] F. Eigner and M. Maffei. Differential privacy by typing in security protocols. In *Proceedings of CSF'13*, pages 272–286, 2013.

[14] D. Garg, L. Jia, and A. Datta. Policy auditing over incomplete logs: theory, implementation and applications. In *Proceedings of CCS'08*, pages 151–162, 2011.

[15] M. Hennessy, J. Rathke, and N. Yoshida. safedpi: a language for controlling mobile code. *Acta Informatica*, 42(4-5):227–290, 2005.

[16] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120, 2002.

[17] N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1999.

[18] D. Kouzapas and A. Philippou. A typing system for privacy. In *Proceedings of SEFM Workshops 2013*, LNCS 8368, pages 56–68, 2014.

[19] Y. Liu, S. Müller, and K. Xu. A static compliance-checking framework for business process models. *IBM Systems Journal*, 46(2):335–362, 2007.

[20] M. J. May, C. A. Gunter, and I. Lee. Privacy apis: Access control techniques to analyze and verify legal privacy policies. In *Proceedings of CSFW-06*, pages 85–97, 2006.

[21] Q. Ni, E. Bertino, and J. Lobo. An obligation model bridging access control policies and privacy policies. In *Proceedings of SACMAT'08*, pages 133–142, 2008.

[22] H. Nissenbaum. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press, 2010.

[23] B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.

[24] O. Sokolsky, U. Sammapun, I. Lee, and J. Kim. Run-time checking of dynamic properties. *Electronic Notes Theoretical Computer Science*, 144(4):91–108, 2006.

[25] D. J. Solove. A Taxonomy of Privacy. *University of Pennsylvania Law Review*, 154(3):477–560, 2006.

[26] M. C. Tschantz and J. M. Wing. Formal methods for privacy. In *Proceedings of FM'09*, LNCS 5850, pages 1–15. Springer, 2009.