Logical Methods in Computer Science
Vol. 13(4:27)2017, pp. 1–42
www.lmcs-online.org

Submitted Jan. 13, 2016
Published Dec. 20, 2017

PRIVACY BY TYPING IN THE π-CALCULUS

DIMITRIOS KOUZAPAS [a] AND ANNA PHILIPPOU [b]

[a] Department of Computing Science, University of Glasgow
e-mail address: Dimitrios.Kouzapas@gla.ac.uk

[b] Department of Computer Science, University of Cyprus
e-mail address: annap@cs.ucy.ac.cy

Abstract. In this paper we propose a formal framework for studying privacy in information systems. The proposal follows a two-axes schema where the first axis considers privacy as a taxonomy of rights and the second axis involves the ways an information system stores and manipulates information. We develop a correspondence between the schema above and an associated model of computation. In particular, we propose the Privacy calculus, a calculus based on the π-calculus with groups extended with constructs for reasoning about private data. The privacy requirements of an information system are captured via a privacy policy language. The correspondence between the privacy model and the Privacy calculus semantics is established using a type system for the calculus and a satisfiability definition between types and privacy policies. We deploy a type preservation theorem to show that a system respects a policy and it is safe if the typing of the system satisfies the policy. We illustrate our methodology via analysis of two use cases: a privacy-aware scheme for electronic traffic pricing and a privacy-preserving technique for speed-limit enforcement.

1. Introduction

The notion of privacy is a fundamental notion for society and, as such, it has been an object of study within various disciplines such as philosophy, politics, law, and culture. In general terms, an analysis of privacy reveals a dynamic concept strongly dependent on cultural norms and evolution. Society today is evolving steadily into an information era where computer systems are required to aggregate and handle huge volumes of private data. Interaction of individuals with such systems is expected to reveal new limits of tolerance towards what is considered private which in turn will reveal new threats to individual privacy. But, fortunately, along with the introduction of new challenges for privacy, technology can also offer solutions to these new challenges.

1.1. A Formal methods approach to privacy.

While the technological advances and the associated widespread aggregation of private data are rendering the need for developing systems that preserve individual's privacy increasingly important, the established techniques for providing guarantees that a system handles information in a privacy-respecting manner are reported as partial and unsatisfactory.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.23638/LMCS-13(4:27)2017

To this effect, the motivation of this paper is to address this challenge by developing formal frameworks for reasoning about privacy-related concepts. Such frameworks may provide solid foundations for understanding the notion of privacy and allow to formally model and study privacy-related situations. Rigorous analysis of privacy is expected to give rise to numerous practical frameworks and techniques for developing sound systems with respect to privacy properties. More specifically, a formal approach to privacy may lead to the development of programming semantics and analysis tools for developing privacy-respecting code. Tools may include programming languages, compilers and interpreters, monitors, and model checkers. Simultaneously, we envision that existing techniques, like privacy by design [34], and proof carrying code [37], may benefit from a formal description of privacy for information systems and offer new and powerful results.

The main objective of this paper is to develop a type-system method for ensuring that a privacy specification, described as a privacy policy, is satisfied by a computational process. At the same time, the paper aims to set the foundations of a methodology that will lead to further research on privacy in information systems. For this reason we define the notion of a privacy model as an abstract model that describes the requirements of privacy in different scenarios and we establish a correspondence between the privacy model and the different abstraction levels inside a computational framework: we show how this privacy model is expressed in terms of programming semantics and in terms of specifying and checking a privacy specification against a program.

1.2. **Contribution.** More concretely the contributions of this paper are:

Privacy Model: In Section 2 we identify a privacy model based on literature by legal scholars [43, 45]. The model is based on a taxonomy of privacy violations that occur when handling private information.

Syntax and Semantics: In Section 3 we propose a variant of the π-calculus with groups [10] to develop a set of semantics that can capture the basic notions of the model we set.

Privacy Policy: We formally describe a privacy requirement as an object of a privacy policy language. The language is expressed in terms of a formal syntax defined in Section 4.

Policy Satisfiability: The main technical contribution of the paper is a correspondence between the privacy policy and π-calculus systems using techniques from the type-system literature for the π-calculus. The correspondence is expressed via a satisfiability definition (Definition 6.9), where a well-typed system satisfies a privacy policy if the type of the system satisfies the policy.

Soundness and Safety: The main results of our approach declare that:

- Satisfiability is sound; it is preserved by the semantics of our underlying computation model (Theorem 6.6).
- A system that satisfies a policy is safe: it will never violate the policy (Theorem 6.14), where violations are expressed as a class of error systems defined with respect to privacy policies (Definition 6.12).

Use cases: We use the results above to develop two real use cases of systems that handle private information and at the same time protect the privacy of the information against external adversaries. Section 7.1 describes the case where an electronic system computes the toll fees of a car based on the car's GPS locations. Section 7.4 describes the case where an authority identifies a speeding car based on the registration number of the car.

2. Overview of the Approach - Methodology

In this section we give an overview of our approach. We begin by discussing a model for privacy which is based on a taxonomy of privacy violations proposed in [43]. Based on this model we then propose a policy language and a model of computation based on the π-calculus with groups so that privacy specifications can be described as a privacy policy in our policy language and specified as terms in our calculus.

2.1. **A Model for Privacy.**

As already discussed, there is no absolute definition of the notion of privacy. Nevertheless, and in order to proceed with a formal approach to privacy, we need to identify an appropriate model that can describe the basics of privacy in the context of information systems. In general terms, privacy can be considered as a set of dynamic relations between individuals that involve privacy rights and permissions, and privacy violations.

2.1.1. *Privacy as a Taxonomy of Rights.*

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 [45] where the authors follow in their arguments the analysis of David Solove, a legal scholar who has provided a discussion of privacy as a taxonomy of possible privacy violations [43]. The privacy model proposed by Solove requires that a *data holder* is responsible for the privacy of a *data subject* against violations from external *adversaries*.

<table>
<tr><td rowspan="2">Invasion</td><td colspan="3">Information</td></tr>
<tr><td>Collection</td><td>Processing</td><td>Dissemination</td></tr>
<tr><td>Intrusion</td><td>Surveillance</td><td>Aggregation</td><td>Breach of Confidentiality</td></tr>
<tr><td>Decisional Interference</td><td>Interrogation</td><td>Identification</td><td>Disclosure</td></tr>
<tr><td rowspan="4"></td><td rowspan="4"></td><td>Insecurity</td><td>Exposure</td></tr>
<tr><td rowspan="2">Secondary Use</td><td>Increased Accessibility</td></tr>
<tr><td>Exclusion</td><td>Blackmail</td></tr>
<tr><td></td><td>Appropriation</td></tr>
<tr><td></td><td></td><td></td><td>Distortion</td></tr>
</table>

Figure 1: A taxonomy on privacy violations

According to Solove, and based on an in-depth study of privacy within the legal field, privacy violations can be distinguished in four categories as seen in Figure 1: i) *invasions*; ii) *information collection*; iii) *information processing*; and iv) *information dissemination*. Invasion-related privacy violations are violations that occur on the physical sphere of an individual and are beyond the context of computer systems. Information-related privacy violations, on the other hand, are concerned with the manipulation of an individual's personal information in ways that may cause the individual to be exposed, threatened, or feel uncomfortable as to how this personal information is being used.

In Figure 1 we can see the taxonomy developed by Solove. We concentrate the discussion on the latter three information-related categories. In the category of *information collection* we have the privacy violations of *surveillance* and *interrogation*. Surveillance has to do with the collection of information about individuals without their consent, e.g., by eavesdropping on communication channels in systems. Interrogation puts the data subject in the awkward position of denying to answer a question.

The second category has to do with *information-processing* violations. The first violation is the one of *aggregation* which describes the situation where a data holder aggregates information about a data subject: while a single piece of information about an individual may not pose a threat to the individual's privacy, a collection of many pieces of information may reveal a deeper understanding about the individual's character and habits. The violation of *identification* occurs when pieces of anonymous information are matched against information coming from a known data subject. This may take place for example by matching the information of data columns between different tables inside a database and may lead to giving access to private information about an individual to unauthorized entities. *Insecurity* has to do with the responsibility of the data holder against any sensitive data of an individual. Insecurity may lead to identity theft, identification or, more generally, bring individuals to harm due to their data being not sufficiently secure against outside adversaries. For example, in the context of information systems passwords should be kept secret. *Secondary usage* arises when a data holder uses the data of an individual for purposes other than the ones the individual has given their consent to. For example, a system that records economic activity for logistic reasons uses the stored data for marketing. The next violation relates to the right of data subjects to access their private data and be informed regarding the reasons and purposes the data are being held. In the opposite case we identify the violation of *exclusion*.

The last category is *information dissemination*. Private information should be disseminated under conditions. If not, we might have the violations of *breach of confidentiality*, *disclosure* and *exposure*. In the first case we are concerned with confidential information. In the second case we assume a non-disclosure agreement between the data holder and the data subject, whereas the third case is concerned with the exposure of embarrassing information about an individual. Following to the next violation, *increased accessibility* may occur when an adversary has access to a collection of non-private pieces of information but chooses to make the collection of these data more widely available. For example, while an email of an employee in a business department may be a piece of public information, publishing a list of such emails constitutes an increased-accessibility violation as it may be used for other reasons such as advertise spam. The violation of *blackmail* occurs when an adversary threatens to reveal private information. *Appropriation* occurs when an adversary associates private information with a product and without the consent of the data subject, and, finally, *distortion* involves the dissemination of false information about an individual that may harm the way the individual is being judged by others.

2.1.2. *A Privacy Model for Information Systems.* Based on the discussion above we propose the following model for privacy for information systems: An information system, the *data holder*, is responsible for the privacy of data of a *data subject* against violations from various *adversaries* which can be users of the modules of the system or external entities. These adversaries may perform operations on private data, e.g., store, send, receive, or process the data and, depending on these operations, privacy violations may occur. For example,

sending private data from a data holder module to an unauthorised module may result in the violation of disclosure.

At the centre of this schema we have the notion of *private data*. In our model, we take the view that private data are data structures representing pieces of information related to individuals along with the identities of the associated individuals. For example, an on-line company may store in its databases the address, wish list, and purchase history for each of its customers.

Viewing in our model private data as associations between individuals and pieces of information is useful for a number of reasons. To begin with this approach allows us to distinguish between private data and other pieces of data. For instance, a certain address on a map has no private dimension until it is defined to be the address of a certain individual. Furthermore, considering private data to be associations between information and individuals enables us to reason about a variety of privacy features. One such feature is anonymisation of private data which occurs when the identity of the individual associated with the data is stripped away from the data. Similarly, identification occurs when one succeeds in filling in the identity associated with a piece of anonymised data. Moreover, aggregation of data is considered to take place when a system collects many pieces of information relating to the same individual. Given the above, in our model we consider private data as a first-class entity, and we make a distinction between private data and other pieces of data.

Taking a step further, in our model we distinguish between different types of private data. This is the case in practice since, by nature, some data are more sensitive than others and compromising it may lead to different types of violations. For example, mishandling the number of a credit card may lead to an identity-theft violation while compromising a social security number may lead to an identification violation. Similarly, we make a distinction between the different entities of an information system, based on the observation that different entities may be associated with different permissions with respect to different types of private data.

Finally, a model of privacy should include the notion of a *purpose*. As indicated in legal studies, it is often the case that private data may be used by a data holder for certain purposes but not for others. As such, the notion of privacy purpose has been studied in the literature, on one hand, in terms of its semantics [44], and, on the other hand, in terms of policy languages and their enforcement [9, 12, 29]. In our model, we encompass the notion of a purpose in terms of associating data manipulation with purposes.

2.2. **Privacy Policies.** After identifying a model for privacy, the next step is to use a proper *description language* able to describe the privacy model and serve as a bridge between the privacy terminology and a formal framework.

To achieve this objective, we create a policy language that enables us to describe privacy requirements for private data over data entities. For each type of private data we expect entities to follow different policy requirements. Thus, we define policies as objects that describe a hierarchical nesting of entities where each node/entity of the hierarchy is associated with a set of privacy permissions. The choice of permissions encapsulated within a policy language is an important issue because identification of these permissions constitutes, in a sense, a characterization of the notion of privacy. In this work, we make an attempt of identifying some such permissions, our choice emanating from a class of privacy violations of Solove's taxonomy which we refine by considering some common applications where privacy plays a central role.

Example 2.1. As an example consider the medical system of a hospital (Hospital) obligated to protect patients' data (patient_data). Inside the hospital there are five types of adversaries - the database administrator, nurses, doctors, a research department and a laboratory - each implementing a different behaviour with respect to the privacy policy in place. Without any formal introduction, we give the privacy policy for the patient private data as an entity patient_data $\gg H$ where H is a nested structure that assigns different permissions to each level of the hierarchy, as shown below:

patient_data $\gg$ Hospital{}[
DBase{store, aggregate},
Nurse{reference, disseminate Hospital ∞},
Doctor{reference, read, update, readId, usage{diagnosis}, no dissemination⟨confidential⟩}
Research{reference, read, usage{research}, no dissemination⟨disclosure⟩}
Lab{reference, read, readId, identify{crime}, disseminate Police 1}
]

According to the policy the various adversaries are assigned permissions as follows: A database administrator (DBase) has the right to store (store) and aggregate (aggregate) patients' data in order to compile patients' files. A nurse (Nurse) in the hospital is able to access (reference) a patient's file but not read or write data on the file. A nurse may also disseminate copies of the file inside the hospital (disseminate Hospital ∞), e.g., to a Doctor. A doctor (Doctor) in the Hospital may gain access (reference) to a patient's file, read it (read) with access to the identity of the patient (readId), and update (update) the patient's information. A doctor may also use the patient's data to perform a diagnosis (usage{diagnosis}) but cannot disseminate it since this would constitute a breach of confidentiality (no dissemination⟨confidential⟩).

In turn, a research department (Research) can access a patient's file (reference), read the information (read), and use it for performing research (usage{research}). However, it is not entitled to access the patient's identity (lack of readId permission) which implies that all information available to it should be anonymised. The research department has no right of disclosing information (no dissemination⟨disclosure⟩). Finally, a laboratory within the hospital system (Lab) is allowed to gain access and read private data, including the associated identities (reference, read, readId) and it may perform identification using patient data against evidence collected on a crime scene (permission identify{crime}). If an identification succeeds then the Lab may disseminate the patient's identity to the police (permissions disseminate Police 1).

2.3. **The π-calculus with groups.** Moving on to the framework underlying our study, we employ a variant of the π-calculus with groups [10] which we refer to as the Privacy calculus. The π-calculus with groups extends the π-calculus with the notion of *groups* and an associated type system in a way that controls how data are being processed and disseminated inside a system. It turns out that groups give a natural abstraction for the representation of entities in a system. Thus, we build on the notion of a group of the calculus of [10], and we use the group memberships of processes to distinguish their roles within systems.

To better capture our distilled privacy model, we extend the π-calculus with groups as follows. To begin with, we extend the calculus with the notion of private data: as already discussed, we take the view that private data are structures of the form $\mathsf{id} \otimes a$ where id is the identity of an individual and a is an information associated with the individual. To indicate private data whose identify is unknown we write $\_ \otimes a$. Furthermore, we define the notion of a *store* which is a process that stores and provides access to private data. Specifically, we

write $\overline{r} \triangleright [\mathsf{id} \otimes a]$ for a store containing the private data $\mathsf{id} \otimes a$ with r being a link/reference via which this information may be read or updated. As we show, these constructs are in fact high-level constructs that can be encoded in the core calculus, but are useful for our subsequent study of capturing and analyzing privacy requirements.

Given this framework, information collection, processing and dissemination issues can be analysed through the use of names/references of the calculus in input, output and object position to identify when a channel is reading, writing or otherwise manipulating private data, or when links to private data are being communicated between groups.

Example 2.2. An implementation of the hospital scenario in Example 2.1 in the π-calculus with groups could be as follows:

$$
\begin{array}{l}
\mathsf{Hospital}[\\
\quad \mathsf{DBase}[\ \overline{r_1} \triangleright [\mathsf{id} \otimes \mathsf{dna}] \mid \overline{r_2} \triangleright [\mathsf{id} \otimes \mathsf{medication}]\ ] \\
\quad \parallel\ \mathsf{Nurse}[\ a!\langle r_1, r_2\rangle.\, \mathbf{0}\ ] \\
\quad \parallel\ \mathsf{Nurse}[\ b!\langle r_1\rangle.\, b!\langle r_1\rangle.\, \mathbf{0}\ ] \\
\quad \parallel\ \mathsf{Doctor}[\ a?(w,z).\, w?(x \otimes y).\, \mathtt{if}\ y = \mathsf{disease}_1\ \mathtt{then}\ z!\langle x \otimes \mathsf{medication}'\rangle.\, \mathbf{0}\ \mathtt{else}\ \mathbf{0}\ ] \\
\quad \parallel\ \mathsf{Research}[\ b?(w).\, w?(\_ \otimes y).\, \mathtt{if}\ y = \mathsf{disease}_2\ \mathtt{then}\ P\ \mathtt{else}\ Q\ ] \\
\quad \parallel\ \mathsf{Lab}[\ b?(w).\, w?(x \otimes y).\, r?(\_ \otimes z).\, \mathtt{if}\ y = z\ \mathtt{then}\ c!\langle w\rangle.\, \mathbf{0}\ \mathtt{else}\ \mathbf{0}\ ] \\
]
\end{array}
$$

In this system, Hospital constitutes a group that is known to the six processes of the subsequent parallel composition. The six processes are defined on groups DBase, Nurse Doctor, Research, and Lab nested within the Hospital group. The group memberships of the processes above characterise their nature and allow us to endow them with permissions while reflecting the entity hierarchy expressed in the privacy policy defined above.

The system above describes the cases where:

- A DBase process defines *store* processes that store the patient data dna and medication associated with a patient's identity, with dna corresponding to the dna of the patient and medication to the current treatment of the patient.
- A Nurse process may hold a patient's files, represented with names r_1 and r_2, and can disseminate them inside the Hospital. In this system there are two Nurse processes that, respectively, disseminate patient's files to the doctor via channel a, or to the labs via channel b (the patient file is represented by reference r_1).
- A Doctor may receive a patient's file, read it, and then, through the conditional statement, use it to perform diagnosis. As we will see below, we identify diagnosis through the type of the matching name $\mathsf{disease}_1$ which in this scenario represents the genetic fingerprint of a disease. After the diagnosis the Doctor may update the treatment of the patient in the corresponding store.
- A Research lab may receive a patient's file and perform statistical analysis based on the matching of the genetic material of the patient. The patient's data made available to the research lab is anonymised, as indicated by the private-data value $\_ \otimes z$.
- A Lab may receive a patient's file, and perform forensic analysis on the patient's dna $(x \otimes y)$ to identify dna evidence connected to a crime $(\_ \otimes z)$. After a successful identification the Lab may inform the Police via channel c.

The implementation above can be associated to the privacy policy defined earlier using static typing techniques for the π-calculus. We clarify this intuition by an informal analysis of the Doctor process. Without any formal introduction to types, assume that structure $x \otimes y$

has type patient_data[dna] that concludes that variable w, being a channel that disseminates such information within the Hospital system, has type Hospital[patient_data[dna]]. Structure $\mathsf{disease}_1$ has type diagnosis[dna] indicating that it is a constant piece of information that can be used for the purposes of performing a diagnosis. Furthermore, structure $x \otimes \mathsf{medication}'$ has type patient_data[treatment]. The fact that the Doctor receives the patient's file on name w signifies the reference permission, whereas inputting value $x \otimes y$ signifies the read permission. The matching operator between patient_data and diagnosis data identifies the usage of private data for the purpose of a diagnosis. Finally, the Doctor has the right to update private data on the patient's file on name z. We can see that the Doctor does not disseminate the file (names w and z) outside its group, thus, there is respect of the $\mathsf{no\ dissemination}\langle\mathsf{confidential}\rangle$ permission.

Intuitively, we may see that this system conforms to the defined policy, both in terms of the group structure as well as the permissions exercised by the processes. Instead, if the nurse were able to engage in a $r_1?x \otimes y$ action then the defined policy would be violated because the nurse would have read the patient's private data without having the permission to do so. Thus, the encompassing group is essential for capturing requirements of non-disclosure.

Using these building blocks, our methodology is applied as follows: Given a system and a typing we perform type checking to confirm that the system is well-typed while we infer a permission interface. This interface captures the permissions exercised by the system. To check that the system complies with a privacy policy we provide a correspondence between policies and permission interfaces the intention being that: a permission interface satisfies a policy if and only if the system exercises a subset of the allowed permissions of the policy. With this machinery at hand, we state and prove a safety theorem according to which, if a system Sys type-checks against a typing $\Gamma; \Lambda$ and produces an interface Θ, and Θ satisfies a privacy policy $\mathcal{P}$, then Sys respects policy $\mathcal{P}$.

3. Calculus

In this section we define a model of concurrent computation whose semantics captures the privacy model we intend to investigate. The calculus we propose is called the Privacy calculus and it is based on the π-calculus with groups originally presented by Cardelli et al. [10]. The π-calculus with groups is an extension of the π-calculus with the notion of a group and an operation of group creation, where a group is a type for channels. In [10] the authors establish a close connection between group creation and secrecy as they show that a secret belonging to a certain group cannot be communicated outside the initial scope of the group.

The Privacy calculus extends the π-calculus with groups with some additional constructs that are useful for our privacy investigation. In fact, as we show, these additional constructs are high-level operations that can be encoded in the core calculus. We begin our exposition by describing the basic intuition behind the design choices of the calculus:

(1) Private data are typically data associated to individuals. For instance, an election vote is not a private piece of information. It becomes, however, private when associated with an identifying piece of information such as a name, a social security number, or an address. So, in the Privacy calculus we distinguish between π-calculus names and private data. In particular, we assume that *private data* are structures that associate constants, that is, pieces of information, with identifying pieces of information which we simply refer to as *identities*.

$$
\begin{array}{lcll}
\text{(identity values)} & \iota & ::= & \mathsf{id} \mid \_ \mid x \\
\text{(data values)} & \delta & ::= & c \mid x \\
\\
\text{(private data)} & \iota \otimes \delta & & \text{where } \iota \neq x \implies \delta = c \text{ and } \iota = x \implies \delta = y \\
\\
\text{(identifiers)} & u & ::= & a \mid r \mid x \\
\text{(terms)} & t & ::= & a \mid r \mid \iota \otimes \delta \mid c \mid x \\
\text{(constant terms)} & v & ::= & a \mid r \mid \mathsf{id} \otimes c \mid c \\
\text{(placeholders)} & k & ::= & x \mid x \otimes y \mid \_ \otimes x \\
\\
\text{(processes)} & P & ::= & \mathbf{0} \mid u!\langle t \rangle.\, P \mid u?(k).\, P \mid (\nu\ n)P \mid P \,|\, P \mid *P \\
 & & \mid & \mathtt{if}\ t = t\ \mathtt{then}\ P\ \mathtt{else}\ P \mid \overline{r} \triangleright [\iota \otimes \delta] \\
\\
\text{(systems)} & S & ::= & \mathsf{G}[P] \mid \mathsf{G}[S] \mid S \| S \mid (\nu\ n)S
\end{array}
$$

Figure 2: Syntax of the Privacy calculus

(2) Privacy enforcement in an information system is about controlling the usage of private data which is typically stored within a database of the system. In the Privacy calculus we capture such databases of private data as a collection of *stores*, where a store is encoded as a high-level process term and includes operations for manipulation of private data. This manipulation takes place with the use of a special kind of names called *references*.

To make the intuition above concrete, let us assume a set of names $\mathcal{N}$, ranged over by $n, m, \ldots$, and partitioned into a set of channel names Ch, ranged over by $a, b, \ldots$, a set of reference names $\mathcal{R}$, ranged over by $r, r', \ldots$, and a set of dual endpoints, $\overline{\mathcal{R}}$, where for each reference $r \in \mathcal{R}$ we assume the existence of a dual endpoint $\overline{r} \in \overline{\mathcal{R}}$. The endpoint $\overline{r}$ belongs solely to a store-term and is used for providing access to the associated private data whereas the endpoint r is employed by processes that wish to access the data. Note that the main distinguishing feature between the two endpoints of a reference is that an endpoint $\overline{r}$ cannot be passed as an object of a communication, whereas an endpoint r can: while it is possible to acquire a reference for accessing an object during computation it is not possible to acquire a reference for providing access to a store. Finally, we point out that channels do not require dual endpoints so we assume that $\overline{a} = a$.

In addition to the set of names our calculus makes use of the following entities. We assume a set of groups [10], $\mathcal{G}$, that ranges over $\mathsf{G}, \mathsf{G}_1, \ldots$. Groups are used to control name creation and to provide the property of secrecy for the information that is being exchanged while characterising processes within a system. Furthermore, we assume a set of variables $\mathcal{V}$ that ranges over $w, x, y, z, \ldots$. Data are represented using the constants set $\mathcal{C}$ ranged over by c, while identities id range over a set of identities Id. We assume the existence of a special identity value '$\_$' called the hidden identity that is used to signify that the identity of private data is hidden. A hidden identity is used by the calculus to enforce private data anonymity.

The syntax of the Privacy calculus is defined in Figure 2. We first assume that an *identity value* ι can be an identity id, a hidden identity $\_$ or an identity variable x. We also define a *data value* δ to be a constant c or a variable x.

As already discussed, private data are considered to be structures that associate an identity value with a data value, written $\iota \otimes \delta$. Structure $\mathsf{id} \otimes c$ associates information c with identity id, while structure $\_ \otimes c$ contains private information c about an individual without revealing the individual's identity. Finally, private data can take one of the forms $x \otimes y$ and $\_ \otimes x$. Note that private data are restricted by definition only to the four forms defined above. Any other form of $\iota \otimes \delta$, e.g., $\mathsf{id} \otimes x$, is disallowed by definition.

Based on the above, the meta-variables of the calculus include the following:

- *identifiers* u denote channels, references or variables,
- *terms* t are channels, references, private data, constants or variables,
- *constant terms* v include all terms except variables, and
- *placeholders* k describe either variables or variable private data.

The syntax of the calculus is defined at two levels, the process level, P, and the system level, S. At the *process* level we have the π-calculus syntax extended with the syntax for stores. Process $\mathbf{0}$ is the inactive process. The output prefix $u!\langle t\rangle.\, P$ denotes a process that sends a term t over identifier u and proceeds as P. As already mentioned, term t may not be the dual endpoint of a reference. Dually, the input prefix $u?(k).\, P$ denotes a process that receives a value over identifier u, substitutes it on variable placeholder k, and proceeds as P. Process $(\nu\ n)P$ restricts name n inside the scope of P. Process $P \mid Q$ executes processes P and Q in parallel. Process $*P$ can be read as an infinite number of P's executing in parallel. The conditional process $\mathtt{if}\ t_1 = t_2\ \mathtt{then}\ P\ \mathtt{else}\ Q$ performs matching on terms t_1 and t_2 and continues to P if the match succeeds, and to Q otherwise. Finally, process $\overline{r} \triangleright [\iota \otimes \delta]$ is a process that is used to store data of the form $\mathsf{id} \otimes c$. We point out that store references cannot be specified via variable placeholders. This is to ensure that each store has a unique store reference. Nonetheless, stores can be created dynamically, using the construct of name restriction in combination with process replication.

Free and bound variables of a process P, denoted $\mathsf{fv}(P)$ and $\mathsf{bv}(P)$, respectively, follow the standard π-calculus definition for all π-calculus constructs, whereas for store processes we define $\mathsf{bv}(\overline{r} \triangleright [\iota \otimes \delta]) = \emptyset$ and $\mathsf{fv}(\overline{r} \triangleright [x \otimes y]) = \{x, y\}$. Additionally, it is convenient to define $\mathsf{fv}(x) = \{x\}$ and $\mathsf{fv}(x \otimes y) = \{x, y\}$. The substitution function, denoted $\{^{v}/_{k}\}$, is then defined as follows: In the case that k is x then, the substitution follows the standard π-calculus definition and results in substituting all free occurrences of k by v with renaming of bound names, if necessary, to prevent any occurrences of v from becoming bounded. In the case that $k = x \otimes y$ and $v = \mathsf{id} \otimes c$, the substitution results in substituting all free occurrences of x and y by id and c, respectively. Finally, in the case that $k = \_ \otimes y$ and $v = \_ \otimes c$, we define $\{^{v}/_{k}\}$ as the substitution of all free occurrences of y by c. Note that the substitution function is undefined for other combinations of v and k.

In turn, systems are essentially processes that are extended to include the group construct. Groups are used to arrange processes into multiple levels of naming abstractions. The group construct is applied both at the level of processes $\mathsf{G}[P]$ and at the level of systems $\mathsf{G}[S]$. Finally, we have the name restriction construct as well as parallel composition for systems.

Note that, for the sake of simplicity, we have presented the monadic version of the Privacy calculus. The calculus together with the type system can be extended in the polyadic case in a straightforward manner. Furthermore, we point out that, unlike the π-calculus with groups, the Privacy calculus does not include a group creation construct. This is due to the fact that groups in our framework are associated with classes of entities and their associated privacy permissions. Thus, the groups of a system need to be fixed.

3.1. **Labelled Transition Semantics.** We present the semantics of the Privacy calculus through a labelled transition system (LTS). We define a labelled transition semantics instead of a reduction semantics due to a characteristic of the intended structural congruence in the Privacy calculus. In particular, the definition of such a congruence would omit the axiom $\mathsf{G}[S_1 \mid S_2] \equiv \mathsf{G}[S_1] \mid S_2$ if $\mathsf{G} \notin \mathtt{fg}(S_2)$ as it was used in [10]. 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 $\mathsf{G}[P]$, cannot be lost even if $\mathsf{G} \notin \mathtt{fg}(P)$, since the $\mathsf{G}[\cdot]$ construct has the additional meaning of group membership in the Privacy calculus and it instils P with privacy-related permissions as we will discuss in the sequel. The absence of this law renders the reduction semantics rule of parallel composition rather complex.

To define a labelled transition semantics we first introduce the set of labels:

$$\ell \quad ::= \quad (\nu\ \tilde{m})n!\langle v\rangle \quad \mid \quad n?(v) \quad \mid \quad \tau$$

We distinguish three action labels. The output label $(\nu\ \tilde{m})n!\langle v\rangle$ denotes an output action on name n that carries object v. Names $\tilde{m}$ is a (possibly empty) set of names in the output object that are restricted. The input label $n?(v)$ denotes the action of inputting value v on name n. Action τ is the standard internal action. To clarify internal interaction we define a symmetric duality relation dual over labels ℓ:

$$(\nu\ \tilde{m})a!\langle v\rangle\ \mathsf{dual}\ a?(v)$$

$$\frac{(v_1 = v_2) \vee (v_1 = \mathsf{id} \otimes c \wedge v_2 = \_ \otimes c)}{(\nu\ \tilde{m})\overline{r}!\langle v_1\rangle\ \mathsf{dual}\ r?(v_2)} \qquad \frac{(v_1 = v_2) \vee (v_2 = \mathsf{id} \otimes c \wedge v_1 = \_ \otimes c)}{(\nu\ \tilde{m})r!\langle v_1\rangle\ \mathsf{dual}\ \overline{r}?(v_2)}$$

The first pair of the relation defines the standard input/output duality between label actions, where an output on name a matches an input on name a that carries the same object. The last two duality pairs relate actions on dual reference endpoints: we distinguish between the case where dual reference endpoints carry the same object and the case where an input (resp., output) without an identity is matched against an output (resp., input) with an identity. Thus, we allow communicating private data without revealing their identity.

The labelled transition semantics for processes is defined in Figure 3 and the labelled transition semantics for systems in Figure 4.

According to rule [Out], output-prefixed process $n!\langle v\rangle.\, P$ may interact on action $n!\langle v\rangle$ and continue as P. Similarly, input-prefixed process $n?(k).\, P$ may interact on action $n?(v)$ and continue as P with k substituted by v as in rule [Inp]. Naturally, the input action can only take place for compatible v and k values which are the cases when the substitution function is defined. Rules [SOut] and [SInp] illustrate the two possible actions of a store process: a store may use the dual endpoint of the store reference r to engage in action $\overline{r}!\langle \mathsf{id} \otimes c\rangle$ or in action $\overline{r}?(\mathsf{id} \otimes c)$. In the first case it returns to its initial state (rule [SOut]) and in the second case it updates the store accordingly (rule [SInp]). Note that, for the input action, if the store already contains data, then the received data should match the identity of the stored data. In turn, rules [True] and [Fls] define the semantics of the conditional construct for the two cases where the condition evaluates to true and false, respectively. Through rule [Res] actions are closed under the restriction operator provided that the restricted name does not occur free in the action. Rule [Scp] extends along with the action $(\nu\ \tilde{m})n!\langle v\rangle$ the scope of name m if m is restricted. Next, rule [Tau] captures that parallel processes may communicate with each other on dual actions and produce action τ, whereas rules [LPar] and

$$[\mathsf{Out}]\ n!\langle v\rangle.\,P \xrightarrow{n!\langle v\rangle} P \qquad [\mathsf{Inp}]\ n?(k).\,P \xrightarrow{n?(v)} P\{^{v}/_{k}\} \qquad [\mathsf{SOut}]\ \overline{r} \triangleright [\mathsf{id} \otimes c] \xrightarrow{\overline{r}!\langle \mathsf{id} \otimes c\rangle} \overline{r} \triangleright [\mathsf{id} \otimes c]$$

$$[\mathsf{SInp}]\ \frac{\iota = x \vee \iota = \mathsf{id}}{\overline{r} \triangleright [\iota \otimes \delta] \xrightarrow{\overline{r}?(\mathsf{id} \otimes c)} \overline{r} \triangleright [\mathsf{id} \otimes c]} \qquad [\mathsf{True}]\ \frac{P \xrightarrow{\ell} P'}{\mathtt{if}\ a = a\ \mathtt{then}\ P\ \mathtt{else}\ Q \xrightarrow{\ell} P'}$$

$$[\mathsf{Fls}]\ \frac{Q \xrightarrow{\ell} Q' \qquad a \neq b}{\mathtt{if}\ a = b\ \mathtt{then}\ P\ \mathtt{else}\ Q \xrightarrow{\ell} Q'} \qquad [\mathsf{Res}]\ \frac{P \xrightarrow{\ell} P' \qquad n \notin \mathsf{fn}(\ell)}{(\nu\ n)P \xrightarrow{\ell} (\nu\ n)P'}$$

$$[\mathsf{Scp}]\ \frac{P \xrightarrow{(\nu\ \tilde{m})n!\langle v\rangle} P' \qquad m \in \mathsf{fn}(v)}{(\nu\ m)P \xrightarrow{(\nu\ m,\tilde{m})n!\langle v\rangle} P'} \qquad [\mathsf{Tau}]\ \frac{P \xrightarrow{\ell_1} P' \qquad Q \xrightarrow{\ell_2} Q' \qquad \ell_1\ \mathsf{dual}\ \ell_2}{P \mid Q \xrightarrow{\tau} (\nu\ \mathsf{bn}(\ell_1) \cup \mathsf{bn}(\ell_2))(P' \mid Q')}$$

$$[\mathsf{LPar}]\ \frac{P \xrightarrow{\ell} P' \qquad \mathsf{bn}(\ell) \cap \mathsf{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\ell} P' \mid Q} \qquad [\mathsf{RPar}]\ \frac{Q \xrightarrow{\ell} Q' \qquad \mathsf{bn}(\ell) \cap \mathsf{fn}(P) = \emptyset}{P \mid Q \xrightarrow{\ell} P \mid Q'}$$

$$[\mathsf{Repl}]\ \frac{P \xrightarrow{\ell} P'}{*P \xrightarrow{\ell} P' \mid *P} \qquad [\mathsf{Alpha}]\ \frac{P \equiv_\alpha P'' \quad P'' \xrightarrow{\ell} P'}{P \xrightarrow{\ell} P'}$$

Figure 3: Labelled Transition System for Processes

$$[\mathsf{SPGr}]\ \frac{P \xrightarrow{\ell} P'}{\mathsf{G}[P] \xrightarrow{\ell} \mathsf{G}[P']} \qquad [\mathsf{SGr}]\ \frac{S \xrightarrow{\ell} S'}{\mathsf{G}[S] \xrightarrow{\ell} \mathsf{G}[S']} \qquad [\mathsf{SRes}]\ \frac{S \xrightarrow{\ell} S' \qquad n \notin \mathsf{fn}(\ell)}{(\nu\ n)S \xrightarrow{\ell} (\nu\ n)S'}$$

$$[\mathsf{SScp}]\ \frac{S \xrightarrow{(\nu\ \tilde{m})n!\langle v\rangle} S' \qquad m \in \mathsf{fn}(v)}{(\nu\ m)S \xrightarrow{(\nu\ m,\tilde{m})n!\langle v\rangle} S'} \qquad [\mathsf{SLPar}]\ \frac{S_1 \xrightarrow{\ell} S_1' \qquad \mathsf{bn}(\ell) \cap \mathsf{fn}(S_2) = \emptyset}{S_1 \| S_2 \xrightarrow{\ell} S_1' \| S_2}$$

$$[\mathsf{SRPar}]\ \frac{S_2 \xrightarrow{\ell} S_2' \qquad \mathsf{bn}(\ell) \cap \mathsf{fn}(S_1) = \emptyset}{S_1 \| S_2 \xrightarrow{\ell} S_1 \| S_2'} \qquad [\mathsf{STau}]\ \frac{S_1 \xrightarrow{\ell_1} S_1' \qquad S_2 \xrightarrow{\ell_2} S_2' \qquad \ell_1\ \mathsf{dual}\ \ell_2}{S_1 \| S_2 \xrightarrow{\tau} (\nu\ \mathsf{bn}(\ell_1) \cup \mathsf{bn}(\ell_2))(S_1' \| S_2')}$$

Figure 4: Labelled Transition System for Systems

[RPar] are symmetric rules that state that actions are closed under the parallel composition provided that there is no name conflict between the bounded names of the action and the free names of the parallel process. Continuing to the replication operator, [Repl] states that $*P$ may execute an action of P and produce the parallel composition of the replication and the continuation of P, and, according to rule [Alpha], the labelled transition system is closed under alpha-equivalence ($\equiv_\alpha$).

Moving on to the semantics of systems we have the following: According to rules [SPGr] and [SGr], actions of processes and systems are preserved by the group restriction operator. The remaining rules are similar to their associated counter-parts for processes.

We often write $\longrightarrow$ for $\xrightarrow{\tau}$ and $\Longrightarrow$ for $\longrightarrow^*$. Furthermore, we write $\overset{\hat{\ell}}{\Longrightarrow}$ for $\Longrightarrow\xrightarrow{\ell}\Longrightarrow$ if $\ell = \tau$, and $\Longrightarrow$ if $\ell = \tau$. Finally, given $\tilde{\ell} = \ell_1 \dots \ell_n$, we write $\overset{\tilde{\ell}}{\Longrightarrow}$ for $\xrightarrow{\ell_1} \dots \xrightarrow{\ell_n}$.

It is useful to define a structural congruence relation over the terms of the Privacy calculus.

Definition 3.1. Structural congruence, denoted $\equiv$, is defined separately on process terms and system terms.

- Structural congruence for processes is a binary relation over processes defined as the least congruence that satisfies the rules:

$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid \mathbf{0} \equiv P \qquad (\nu\ a)\mathbf{0} = \mathbf{0}$$
$$n \notin \mathsf{fn}(Q) \implies (\nu\ n)P \mid Q \equiv (\nu\ n)(P \mid Q) \qquad (\nu\ n)(\nu\ m)P \equiv (\nu\ m)(\nu\ n)P$$

- Structural congruence for systems is a binary relation over systems defined as the least congruence that satisfies the rules:

$$P \equiv Q \implies \mathsf{G}[P] \equiv \mathsf{G}[Q] \qquad S_1 \| S_2 \equiv S_2 \| S_1 \qquad (S_1 \| S_2) \| S_3 \equiv S_1 \| (S_2 \| S_3)$$
$$n \notin \mathsf{fn}(S_2) \implies (\nu\ n)S_1 \mid S_2 \equiv (\nu\ n)(S_1 \mid S_2) \qquad (\nu\ n)(\nu\ m)S \equiv (\nu\ m)(\nu\ n)S$$

The structural congruence relation preserves the labelled transition semantics:

Theorem 3.2. *Consider processes P and Q, and systems S_1, S_2.*

- *If $P \equiv Q$ and there exist ℓ and P' such that $P \xrightarrow{\ell} P'$ then there exists Q' such that $Q \xrightarrow{\ell} Q'$ and $P' \equiv Q'$.*
- *If $S_1 \equiv S_2$ and there exist ℓ and S_1' such that $S_1 \xrightarrow{\ell} S_1'$ then there exists S_2' such that $S_2 \xrightarrow{\ell} S_2'$ and $S_1' \equiv S_2'$.*

Proof. The proof is an induction over the definition of $\equiv$. For each case of the induction the proof considers the derivation of each action ℓ. □

The syntax and the semantics of the Privacy calculus are encodable in the standard π-calculus with select and branch operations [27]. The encoding enjoys a form of operational correspondence. The details of the translation and operational correspondence can be found in Appendix A.

Example 3.3. As an example consider the case where a system reads private information from a store term. The derivation for an internal transition for the system above is:

$$[\mathsf{SOut}]\ \frac{\overline{r} \triangleright [\mathsf{id} \otimes c] \overset{\overline{r}!\langle \mathsf{id} \otimes c\rangle}{\longrightarrow} \overline{r} \triangleright [\mathsf{id} \otimes c]}{\mathsf{G}_1[\overline{r} \triangleright [\mathsf{id} \otimes c]] \overset{\overline{r}!\langle \mathsf{id} \otimes c\rangle}{\longrightarrow} \mathsf{G}_1[\overline{r} \triangleright [\mathsf{id} \otimes c]]} \qquad [\mathsf{Inp}]\ \frac{r?(\_ \otimes x).\, P \overset{r?(\_ \otimes c)}{\longrightarrow} P\{^{\_ \otimes c}/_{\_ \otimes x}\}}{\mathsf{G}_2[r?(\_ \otimes x).\, P] \overset{r?(\_ \otimes c)}{\longrightarrow} \mathsf{G}_2[P\{^{\_ \otimes c}/_{\_ \otimes x}\}]}$$

$$[\mathsf{Tau}]\ \frac{\overline{r}!\langle \mathsf{id} \otimes c\rangle \ \mathsf{dual}\ r?(\_ \otimes c)}{\mathsf{G}_1[\overline{r} \triangleright [\mathsf{id} \otimes c]] \parallel \mathsf{G}_2[r?(\_ \otimes x).\, P] \xrightarrow{\tau} \mathsf{G}_1[\overline{r} \triangleright [\mathsf{id} \otimes c]] \parallel \mathsf{G}_2[P\{^{\_ \otimes c}/_{\_ \otimes x}\}]}$$

We observe an output action $\overline{r}!\langle \mathsf{id} \otimes c\rangle$ on the store system which is dual to the input action $r?(\_ \otimes c)$ executed by the receiving system. The duality of the two actions ensures the internal transition of the parallel composition of the two systems. The receiving input action has an anonymous identity and thus the received data is substituted anonymously.

4. Privacy Policy

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 case of a negative norm, on *data attributes* which are types of sensitive data within a system, and, in particular, how the various agents, who are referred to by their *roles*, may/may not handle these data.

The notions of an attribute and a role are reflected in our framework via the notions of private data and groups, respectively. Thus, our policy language is defined in such a way as to specify the allowed and disallowed permissions associated with the various groups for each type of private data. To express this we make use of a type system to distinguish between the various types of data within a system and to associate policies to each different type of private data.

Specifically, we assume a set of ground types, ranged over by g, a set of private data types, ranged over by t, and a set of constant types, ranged over by p. Based on the above we define the following set of types:

$$T \quad ::= \quad \mathsf{t}[\mathsf{g}] \;\mid\; \mathsf{p}[\mathsf{g}] \;\mid\; \mathsf{G}[T]$$

Thus, types range over T and include type $\mathsf{t}[\mathsf{g}]$, used to type private data and type $\mathsf{p}[\mathsf{g}]$ used to type constants. Types are also inductively defined using groups $\mathsf{G}[T]$. The intuition behind the notion of group G in $\mathsf{G}[T]$ is that a name $x : \mathsf{G}[T]$ may only be used for communicating data of type T between processes that *belong* to group G. In our privacy setting we map private data of the form $\mathsf{id} \otimes c$ to private data types $\mathsf{t}[\mathsf{g}]$. This signifies the fact that constant c of type g is associated with an identity in a piece of private information ($\mathsf{id} \otimes c$). The value id is implicitly assumed to be an identity without any extra type assignment. As a result, identities cannot be used as objects in communications or for any other type of processing within our calculus.

Example 4.1. As a running example for this section, consider a system that describes some internal modules of a car:

$$\begin{array}{l}\mathsf{Car}[\\ \quad \mathsf{Speedometer}[\,\overline{r} \triangleright [\mathsf{id} \otimes sp] \mid *\,(\nu\ sp)(r!\langle \mathsf{id} \otimes sp\rangle.\,\mathbf{0})\,] \\ \quad \parallel\ \mathsf{SpeedCheck}[\,r?(x \otimes y).\,\mathtt{if}\ y > \mathsf{speedLim}\ \mathtt{then}\ P\ \mathtt{else}\ Q\,] \\ ]\end{array}$$

The internal module defined by group $\mathsf{Speedometer}$ is a module that models the speed of the car. The speed of the car is considered to be private data and it is associated with the identity, id, of the driver of the car. Thus, $\mathsf{id} \otimes sp$ has type $\mathsf{vehicle\_data}[\mathsf{speed}]$, where $\mathsf{vehicle\_data}$ is the type of private data pertaining to the car and speed is a ground type which corresponds to the speed of the car.

The module defined by group $\mathsf{SpeedCheck}$ is responsible for checking when a car exceeds some speed limit. Group $\mathsf{SpeedCheck}$ inputs the current speed of the car (of type $\mathsf{vehicle\_data}[\mathsf{speed}]$) via name r that has type $\mathsf{Car}[\mathsf{vehicle\_data}[\mathsf{speed}]]$ which represents a name within group Car that can receive data of type $\mathsf{vehicle\_data}[\mathsf{speed}]$. Constant $\mathsf{speedLim}$ has type $\mathsf{Limit}[\mathsf{speed}]$ and it denotes a ground value of type speed, decorated with the constant type Limit, signifying that it is a constant that can be used for the purpose of checking satisfaction of the speed limit.

We define privacy policies as an association of permissions to a hierarchy of groups with respect to types of private data. The set of permissions we consider for handling private data is the following:

$$\begin{array}{rcl} \mathsf{prm} & ::= & \mathsf{read} \mid \mathsf{update} \mid \mathsf{reference} \mid \mathsf{disseminate}\ G\ \lambda \mid \mathsf{store} \mid \mathsf{readId} \\ & \mid & \mathsf{no\ dissemination}\langle\mathsf{kind}\rangle \mid \mathsf{usage}\{\mathsf{p}\} \mid \mathsf{identify}\{\mathsf{t}\} \mid \mathsf{aggregate} \\ \lambda & ::= & 1, 2, \ldots \mid \omega \\ \mathsf{kind} & ::= & \mathsf{disclosure} \mid \mathsf{confidential} \mid \mathsf{sensitive} \end{array}$$

Permissions are denoted by prm and are as follows:

- Permission read when associated with a type of private data and a group indicates that the private data may be read by processes belonging to the specific group.
- Permission update gives the possibility of updating the contents of a store with a new piece of private data.
- Permission $\mathsf{reference}$ allows to gain access to a reference on private data.
- Permission $\mathsf{disseminate}\ \mathsf{G}\ \lambda$, when associated with a type of private data, indicates that the private data may be disseminated up to λ times to processes belonging to group G, where λ ranges over natural numbers and ω, with ω being the unlimited number of times.
- Permission store allows to define a store process for the storage of private data.
- Permission readId allows access to the id of the private data. Lack of the readId permission corresponds to the process of anonymisation of private data.
- Permission $\mathsf{no\ dissemination}\langle\mathsf{kind}\rangle$, when associated with a certain type of data and a group, disallows the dissemination of the specific type of data outside the scope of the associated group under the reason described by kind. kind ranges over $\mathsf{disclosure}$ that denotes a non-disclosure agreement, $\mathsf{confidential}$ that denotes a confidentiality relation, and $\mathsf{sensitive}$ that denotes sensitive data.
- Permission $\mathsf{usage}\{\mathsf{p}\}$ defines the right to match private data against constants of type p.
- Permission $\mathsf{identify}\{\mathsf{t}\}$ allows the identification of private data against private data with type t. Typically, we require that t characterises data that are unique for each individual, such as identity numbers and passwords.
- Finally, permission $\mathsf{aggregate}$ grants the right to aggregate private data. Data aggregation takes place when a component in the system hierarchy is allowed to store more than a single piece of private information for the same data subject.

In turn, a policy is an assignment of permissions to a hierarchy of groups with respect to types of sensitive data and it is defined as follows:

$$\begin{array}{rcl} \mathcal{P} & ::= & \mathsf{t} \gg \mathsf{H} \mid \mathsf{t} \gg \mathsf{H}; \mathcal{P} \\ \mathsf{H} & ::= & \mathsf{G}\{\tilde{\mathsf{prm}}\}[\tilde{\mathsf{H}}] \end{array}$$

where $\tilde{\mathsf{prm}}$ is a set of permissions. A policy has the form $\mathsf{t}_1 \gg \mathsf{H}_1; \ldots; \mathsf{t}_n \gg \mathsf{H}_n$ where t_i are the base types subject to privacy. Given policy $\mathcal{P}$ we write $\mathcal{P}(\mathsf{t})$ for H_i where $\mathsf{t} = \mathsf{t}_i$. 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 $\mathsf{G}\{\tilde{\mathsf{prm}}\}[\mathsf{H}_1, \ldots, \mathsf{H}_m]$ which expresses that an entity possessing a group membership in group G has rights $\tilde{\mathsf{prm}}$ to the data in question and, if additionally it is a member of some group G_i, where $\mathsf{H}_i = \mathsf{G}_i\{\tilde{\mathsf{prm}}_i\}[\ldots]$, then it also has the rights $\tilde{\mathsf{prm}}_i$, and so on.

Example 4.2. We define a privacy policy for a network system that utilises the internal car modules of our running example (Example 4.1) as:

$$
\begin{array}{lll}
\mathsf{vehicle\_data} & \gg & \mathsf{Network}\{\}[\\
& & \quad \mathsf{Car}\{\mathsf{disseminate\ Network}\ \omega\}[\\
& & \quad\quad \mathsf{Speedometer}\{\mathsf{update}, \mathsf{store}\}, \\
& & \quad\quad \mathsf{SpeedCheck}\{\mathsf{read}, \mathsf{readId}, \mathsf{usage}\{\mathsf{Limit}\}\} \\
& & \quad], \\
& & \quad \mathsf{SpeedAuthority}\{\mathsf{reference}, \mathsf{read}\} \\
& &]
\end{array}
$$

The privacy requirements of the policy above are concerned with private data of type vehicle_data. We assume that the modules of a car, expressed here by group Car, are connected to a network (Network). Part of the network is a speed authority. The Speedometer module is responsible for displaying the speed of the car. This is modelled by permission store. Permission update enables the Speedometer to regularly update the speed of the car. The SpeedCheck module can read the speed and use it for the purpose of checking it against the speed limit. Any group inside the Car group can disseminate inside group Network a link to the speed store in the Speedometer. This is expressed by permission disseminate Network ω. Finally, the permission reference of the SpeedAuthority allows the SpeedAuthority to get a link to the speed's store. The authority also has permission read so it can read the Speedometer store.

Following the need to combine permission environments, we define the following: Operator $\oplus$ is defined on λ values as the commutative monoid that follows the rules:

$$\lambda \oplus \omega = \omega \qquad\qquad \lambda_1 \oplus \lambda_2 = \lambda_1 + \lambda_2 \quad \lambda_1 \neq \omega, \lambda_2 \neq \omega$$

Operator $\uplus$ is used to combine sets of permissions:

$$
\begin{array}{rcl}
\tilde{\mathsf{prm}}_1 \uplus \tilde{\mathsf{prm}}_2 & = & \{\mathsf{prm} \mid \mathsf{prm} \in \tilde{\mathsf{prm}}_1 \cup \tilde{\mathsf{prm}}_2, \mathsf{prm} \neq \mathsf{disseminate\ G}\ \lambda\} \\
& \cup & \{\mathsf{disseminate\ G}\ (\lambda_1 \oplus \lambda_2) \mid \mathsf{disseminate\ G}\ \lambda_i \in \tilde{\mathsf{prm}}_i, i \in \{1,2\}\} \\
& \cup & \{\mathsf{disseminate\ G}\ \lambda \mid \mathsf{disseminate\ G}\ \lambda \in \mathsf{prm}_1, \mathsf{disseminate\ G}\ \lambda' \notin \mathsf{prm}_2\} \\
& \cup & \{\mathsf{disseminate\ G}\ \lambda \mid \mathsf{disseminate\ G}\ \lambda \in \mathsf{prm}_2, \mathsf{disseminate\ G}\ \lambda' \notin \mathsf{prm}_1\}
\end{array}
$$

Operator $\uplus$ takes the union of two permission sets with the exception of the disseminate G λ permission; whenever we have two disseminate G λ permissions the result is the same permission up-to the addition of the λ value.

We proceed to define auxiliary functions groups(H) and perms(H) so as to gather the sets of groups and the set of permissions, respectively, inside a hierarchy structure:

$$
\begin{array}{rcl}
\mathsf{groups(H)} & = & \{\mathsf{G}\} \cup (\bigcup_{j \in J} \mathsf{groups}(\mathsf{H}_j)) \\
\mathsf{perms(H)} & = & \tilde{\mathsf{prm}} \uplus (\biguplus_{j \in J} \mathsf{perms}(\mathsf{H}_j))
\end{array}
$$

We are now ready to introduce a well-formedness check on the policy structure.

Definition 4.3 (Well-formed Policy)**.** We say that a policy $\mathcal{P} = \mathsf{t}_1 \gg \mathsf{H}_1; \ldots; \mathsf{t}_n \gg \mathsf{H}_n$ is *well formed*, written $\mathcal{P} : \diamond$, if it satisfies the following:

(1) The t_i are distinct.
(2) If $\mathsf{H} = \mathsf{G}\{\tilde{\mathsf{prm}}\}[\mathsf{H}_j]_{j \in J}$ occurs within some H_i then $\mathsf{G} \notin \mathsf{groups}(\mathsf{H}_j)$ for all $j \in J$.

(3) If $\mathsf{H} = \mathsf{G}\{\tilde{\mathsf{prm}}\}[\mathsf{H}_j]_{j\in J}$ occurs within some H_i, $\mathsf{no\ dissemination}\langle\mathsf{kind}\rangle \in \tilde{\mathsf{prm}}$ and $\mathsf{disseminate}\ \mathsf{G}'\ \lambda \in \mathsf{perms}(\mathsf{H}_j)$ for some $j \in J$, then $\mathsf{G}' \in \mathsf{groups}(\mathsf{H})$.

A well-formed policy is required to have definition for distinct private types, an acyclic group hierarchy and, furthermore, no non-disclosure requirement imposed at some level of a hierarchy can be in conflict with a disclosure requirement granted in its sub-hierarchy. Hereafter, we assume that policies are well-formed policies. As a shorthand, we write $\mathsf{G} : \tilde{\mathsf{prm}}$ for $\mathsf{G}\{\tilde{\mathsf{prm}}\}[\epsilon]$ and we abbreviate G for $\mathsf{G} : \emptyset$.

Given a set of groups $\tilde{G}$ and hierarchy H, it is often convenient to extract a flat structure of the policy hierarchy referring solely to the groups $\tilde{G}$. This flat hierarchy captures the nesting of the groups, as defined by the hierarchy, and accumulates the set of permissions associated to agents belonging to the groups in question. Thus, a flat hierarchy has the form

$$\theta \quad ::= \quad \mathsf{G}[\theta] \ \mid \ \mathsf{G}[\tilde{\mathsf{prm}}]$$

Precisely, given a set of groups $\tilde{G}$ and a hierarchy H, we define the flat hierarchy $\mathsf{H}_{\tilde{G}}$ as follows:

$$\mathsf{H}_{\mathsf{G}} = \mathsf{G}[\tilde{\mathsf{prm}}] \quad \text{if} \quad \mathsf{H} = \mathsf{G}\{\tilde{\mathsf{prm}}\}$$
$$(\mathsf{G}\{\tilde{\mathsf{prm}}\}[\tilde{\mathsf{H}}])_{\mathsf{G},\tilde{\mathsf{G}}} = \mathsf{G}[\theta] \quad \text{if} \quad \mathsf{G}'\{\tilde{\mathsf{prm}}'\}[\tilde{\mathsf{H}}'] \in \tilde{\mathsf{H}} \wedge \theta = (\mathsf{G}'\{\tilde{\mathsf{prm}} \uplus \tilde{\mathsf{prm}}'\}[\tilde{\mathsf{H}}'])_{\tilde{\mathsf{G}}}$$

Example 4.4. Returning to our running example, let the policy in Example 4.2 to be $\mathcal{P} = \mathsf{vehicle\_data} \gg \mathsf{H}$. Furthermore, if $\tilde{\mathsf{G}} = \mathsf{Network}, \mathsf{Car}, \mathsf{SpeedCheck}$ then:

$$\mathsf{H}_{\tilde{\mathsf{G}}} = \mathsf{Network}[\mathsf{Car}[\mathsf{SpeedCheck}[\mathsf{disseminate\ Network}\ \omega, \mathsf{read}, \mathsf{readId}, \mathsf{usage}\{\mathsf{Limit}\}]]]$$

The operators $\mathsf{groups}(\cdot)$ and $\mathsf{perms}(\cdot)$ are extended to include θ structures:

$$\mathsf{groups}(\mathsf{G}[\tilde{\mathsf{prm}}]) = \{\mathsf{G}\} \qquad \mathsf{groups}(\mathsf{G}[\theta]) = \{\mathsf{G}\} \cup \mathsf{groups}(\theta)$$
$$\mathsf{perms}(\mathsf{G}[\tilde{\mathsf{prm}}]) = \tilde{\mathsf{prm}} \qquad \mathsf{perms}(\mathsf{G}[\theta]) = \mathsf{perms}(\theta)$$

5. Type System

The goal of this paper is to statically ensure that a process respects a privacy policy. To achieve this we develop a typing system. The key idea is that the type system performs a static analysis on the syntax of processes and derives the behaviour of each process with respect to data collection, data processing, and data dissemination as a type. Then the derived type is checked for satisfaction against a privacy policy.

We partition the typing rules into three categories: i) typing rules for values and expressions; ii) typing rules for processes; and iii) typing rules for systems. We begin by defining the typing environments on which type checking is carried out.

5.1. Typing environments.

We first define the typing environments.

$$\Gamma \quad ::= \quad \Gamma, u : \mathsf{G}[T] \ \mid \ \Gamma, \iota \otimes \delta : \mathsf{t}[\mathsf{g}] \ \mid \ \Gamma, \delta : \mathsf{p}[\mathsf{g}] \ \mid \ \emptyset$$
$$\Delta \quad ::= \quad \Delta, \mathsf{t} : \tilde{\mathsf{prm}} \ \mid \ \emptyset$$

Type environment Γ maps names and constants to appropriate types. Specifically, names u are mapped to types of the form $\mathsf{G}[T]$. References are mapped to private data types of the form $\mathsf{t}[\mathsf{g}]$ and constants are mapped to purpose types of the form $\mathsf{p}[\mathsf{g}]$. Permission

$$[\mathsf{TName}]\ \frac{}{\Gamma, u : T \vdash u : T \triangleright \emptyset} \qquad [\mathsf{TPdata}]\ \frac{}{\Gamma, \iota \otimes \delta : \mathsf{t}[\mathsf{g}] \vdash \iota \otimes \delta : \mathsf{t}[\mathsf{g}] \triangleright \{\mathsf{t} : \mathsf{idperm}(\iota)\}}$$

$$[\mathsf{TCons}]\ \frac{}{\Gamma, \delta : T \vdash \delta : T \triangleright \emptyset} \qquad [\mathsf{TId}]\ \frac{\Gamma \vdash \mathsf{id} \otimes \delta_1 : \mathsf{t}_1[\mathsf{g}] \triangleright \Delta_1 \qquad \Gamma \vdash \_ \otimes \delta_2 : \mathsf{t}_2[\mathsf{g}] \triangleright \Delta_2}{\Gamma \vdash \delta_1 = \delta_2 \triangleright \Delta_1 \uplus \Delta_2 \uplus \{\mathsf{t}_1 : \{\mathsf{identify}\{\mathsf{t}_2\}\}\}}$$

$$[\mathsf{TUse}]\ \frac{\iota \neq \_ \quad \Gamma, \delta_2 : \mathsf{p}[\mathsf{g}] \vdash \iota \otimes \delta_1 : \mathsf{t}[\mathsf{g}] \triangleright \Delta}{\Gamma, \delta_2 : \mathsf{p}[\mathsf{g}] \vdash \delta_1 = \delta_2 \triangleright \Delta \uplus \{\mathsf{t} : \{\mathsf{usage}\{\mathsf{p}\}\}\}}$$

$$[\mathsf{TEqP}]\ \frac{\Gamma \vdash \mathsf{id}_1 \otimes \delta_1 : \mathsf{t}[\mathsf{g}] \triangleright \Delta_1 \qquad \Gamma \vdash \mathsf{id}_2 \otimes \delta_2 : \mathsf{t}[\mathsf{g}] \triangleright \Delta_2}{\Gamma \vdash \mathsf{id}_1 \otimes \delta_1 = \mathsf{id}_2 \otimes \delta_2 \triangleright \Delta_1 \uplus \Delta_2}$$

$$[\mathsf{TEqA}]\ \frac{\Gamma \vdash \_ \otimes \delta_1 : \mathsf{t}[\mathsf{g}] \triangleright \Delta_1 \qquad \Gamma \vdash \_ \otimes \delta_2 : \mathsf{t}[\mathsf{g}] \triangleright \Delta_2}{\Gamma \vdash \_ \otimes \delta_1 = \_ \otimes \delta_2 \triangleright \Delta_1 \uplus \Delta_2}$$

Figure 5: Typing rules for values and expressions

environment Δ assigns a set of privacy permissions to types of private data. Permission environment Δ is intended to be used for conformance against a privacy policy.

Following the need to combine Δ environments, for extracting the interface of parallel processes we extend operator $\uplus$, previously defined for sets of permissions, to permission environments:

$$\Delta_1 \uplus \Delta_2 = \{\mathsf{t} : \tilde{\mathsf{prm}}_1 \uplus \tilde{\mathsf{prm}}_2 \ \mid\ \mathsf{t} : \tilde{\mathsf{prm}}_1 \in \Delta_1, \mathsf{t} : \tilde{\mathsf{prm}}_2 \in \Delta_2\} \cup \Delta_1 \backslash \Delta_2 \ \cup\ \Delta_2 \backslash \Delta_1$$

At the level of permission environments Δ, operator $\uplus$ is in fact the union of the two environments with the exception of common private types where the associated permission sets are combined with the $\uplus$ operator.

5.2. **A type system for values and expressions.** We present the typing rules for values and expressions. Depending on what kind of values and expressions processes use we can derive how a process handles private data.

We use a typing judgement for values v and a typing judgement for the matching expression $v_1 = v_2$:

$$\Gamma \vdash v : T \triangleright \Delta \qquad\qquad \Gamma \vdash v_1 = v_2 \triangleright \Delta$$

Before we define the typing rules we present the auxiliary function $\mathsf{idperm}(\mathsf{id})$:

$$\mathsf{idperm}(\mathsf{id}) = \{\mathsf{readId}\} \qquad \mathsf{idperm}(x) = \{\mathsf{readId}\} \qquad \mathsf{idperm}(\_) = \emptyset$$

The $\mathsf{idperm}(\iota)$ function is used to derive the readId permission out of an id; a visible id maps to the readId permission, while a hidden identity, $\_$, maps to no permissions.

Figure 5 defines the typing rules for values and expressions. Rules [TName], [TCons], and [TPdata] type, respectively, names u, constants δ, and private data $\iota \otimes \delta$ with respect to type environment Γ. Rule [TPdata] is also used to check whether a process has access to the identity of the private data via the definition of $\mathsf{idperm}(\iota)$. Rule for identification [TId] types a matching operation: the key idea is that through matching between data whose identity

is known and data whose identity is unknown, an identification may be performed. For example, if we let:

$$\Gamma = \mathsf{john} \otimes \mathsf{dna}_1 : \mathsf{patient\_data}[\mathsf{DNA}], \_ \otimes \mathsf{dna}_2 : \mathsf{crime}[\mathsf{DNA}]$$

then a forensics lab system defined as $\mathsf{forensics\ lab}[\texttt{if } \mathsf{dna}_1 = \mathsf{dna}_2 \texttt{ then } P \texttt{ else } Q]$ by performing the comparison $\mathsf{dna}_1 = \mathsf{dna}_2$, may identify that the DNA obtained at a crime scene is identical to john's DNA and thus perform an identification for a crime investigation. Thus, the type system, and rule [TId] in particular, will deduce that

$$\Gamma \vdash \mathsf{dna}_1 = \mathsf{dna}_2 \triangleright \mathsf{patient\_data} : \mathsf{identify}\{\mathsf{crime}\}$$

This situation requires that the forensics lab process has permission to identify based on the private data of type patient_data.

Rule [TUse] defines private data usage. We assume that the usage of private data is always reduced to a name matching operation of private data over constant data. For example assume:

$$\Gamma = \mathsf{john} \otimes \mathsf{dna}_1 : \mathsf{patient\_data}[\mathsf{DNA}], \mathsf{dna}_2 : \mathsf{diagnosis}[\mathsf{DNA}]$$

Then a doctor defined as $\mathsf{Doctor}[\texttt{if } \mathsf{dna}_1 = \mathsf{dna}_2 \texttt{ then } P \texttt{ else } Q]$, may use john's patient_data for the purpose of performing a diagnosis. In particular, rule [TUse] allows us to deduce that

$$\Gamma \vdash \mathsf{dna}_1 = \mathsf{dna}_2 \triangleright \mathsf{patient\_data} : \mathsf{usage}\{\mathsf{diagnosis}\}$$

Finally, rules [TEqP] and [TEqA] perform type matching of private data against each other and anonymised data against each other. We take the stance that such comparisons do not exercise any permissions though, we believe, that this is a question for further investigation.

5.3. **A type system for process terms.** Types for processes rely on a linear environment Λ and a store environment Z:

$$\begin{array}{lcl} \Lambda & ::= & \Lambda, r \ \mid \ \emptyset \\ Z & ::= & Z, \langle \iota, \mathsf{t} \rangle \ \mid \ \emptyset \end{array}$$

Environment Λ accumulates the references to stores that are present in a process and its being linear captures that we cannot use the same reference in more than one store. Thus, Λ_1, Λ_2 is defined only when Λ_1 and Λ_2 are disjoint. In turn, environment Z contains the identifiers whose private data is stored in a system along with the respective private type of the stored data. Thus, typing judgements have the form

$$\Gamma; \Lambda; Z \vdash P \triangleright \Delta$$

which essentially states that given a type environment Γ, a reference environment Λ and a store environment, Z, process P is well typed and produces a permission environment Δ.

The typing rules for processes are defined in Figure 6. Rule [TInact] states that the inactive process produces an empty permission environment and rule [TSt] that a store process produces a permission environment which contains the store permission in addition to any further permissions associated with the stored private data.

Rule [TOut] types the output-prefixed process: If environment $\Gamma; \Lambda; Z$ produces Δ_1 as a permission interface of P, u and v have compatible types according to Γ, and v produces a

$$[\mathsf{TInact}]\ \frac{}{\Gamma;\emptyset;Z\vdash \mathbf{0}\triangleright\emptyset} \qquad [\mathsf{TSt}]\ \frac{\Gamma\vdash r:\mathsf{G}[\mathsf{t}[\mathsf{g}]]\triangleright\emptyset \qquad \Gamma\vdash \iota\otimes\delta:\mathsf{t}[\mathsf{g}]\triangleright\Delta'}{\Gamma;r;\langle\iota,\mathsf{t}\rangle\vdash\overline{r}\triangleright[\iota\otimes\delta]\triangleright\mathsf{t}:\{\mathsf{store}\}\uplus\Delta'}$$

$$[\mathsf{TOut}]\ \frac{\begin{array}{c}\Gamma;\Lambda;Z\vdash P\triangleright\Delta_1\\ \Gamma\vdash u:\mathsf{G}[T]\triangleright\emptyset \quad \Gamma\vdash v:T\triangleright\Delta_2\end{array}}{\Gamma;\Lambda;Z\vdash u!\langle v\rangle.\,P\triangleright\Delta_1\uplus\Delta_2\uplus\Delta_T^U} \qquad [\mathsf{TInp}]\ \frac{\begin{array}{c}\Gamma;\Lambda;Z\vdash P\triangleright\Delta_1\\ \Gamma\vdash u:\mathsf{G}[T]\triangleright\emptyset \quad \Gamma\vdash k:T\triangleright\Delta_2\end{array}}{\Gamma\backslash k;\Lambda\backslash k;Z\vdash u?(k).\,P\triangleright\Delta_1\uplus\Delta_2\uplus\Delta_T^R}$$

$$[\mathsf{TRes}]\ \frac{\Gamma;\Lambda;Z\vdash P\triangleright\Delta}{\Gamma\backslash\{n\};\Lambda\backslash\{n\};Z\vdash(\nu\ n)P\triangleright\Delta} \qquad [\mathsf{TRepl}]\ \frac{\begin{array}{c}\Gamma;\emptyset;Z\vdash P\triangleright\Delta\\ \Delta'=\{\mathsf{t}:\{\mathsf{aggregate}\}\ \mid\ \langle\iota,\mathsf{t}\rangle\in Z\}\end{array}}{\Gamma;\emptyset;Z\vdash *P\triangleright\Delta^*\uplus\Delta'}$$

$$[\mathsf{TPar}]\ \frac{\begin{array}{c}\Gamma;\Lambda_i;Z_i\vdash P_i\triangleright\Delta_i \quad i\in\{1,2\}\\ \Delta \ = \ \{\mathsf{t}:\{\mathsf{aggregate}\}\ \mid\\ \qquad \langle\iota,\mathsf{t}\rangle\in Z_i\wedge\langle\iota',\mathsf{t}'\rangle\in Z_j\wedge[\iota=\iota'\vee\iota=x\vee\iota'=x]\wedge\{i,j\}=\{1,2\}\}\end{array}}{\Gamma;\Lambda_1,\Lambda_2;Z_1,Z_2\vdash P_1\mid P_2\triangleright\Delta_1\uplus\Delta_2\uplus\Delta}$$

$$[\mathsf{TIf}]\ \frac{\Gamma;\Lambda_i;Z_i\vdash P_i\triangleright\Delta_i \qquad i\in\{1,2\} \qquad \Gamma\vdash v_1=v_2\triangleright\Delta}{\Gamma;\Lambda_1,\Lambda_2;Z_1,Z_2\vdash \mathtt{if}\ v_1=v_2\ \mathtt{then}\ P_1\ \mathtt{else}\ P_2\triangleright\Delta\uplus\Delta_1\uplus\Delta_2}$$

Figure 6: Typing rules for processes

permission interface Δ_2, we conclude that the process $u!\langle v\rangle.\,P$ produces an interface where the combination of interfaces Δ_1 and Δ_2 is extended with the permissions Δ_T^U, where

$$\Delta_T^U \ = \ \begin{cases}\mathsf{t}:\{\mathsf{update}\} & \text{if} \quad T=\mathsf{t}[\mathsf{g}]\\ \mathsf{t}:\{\mathsf{disseminate\ G\ 1}\} & \text{if} \quad T=\mathsf{G}[\mathsf{t}[\mathsf{g}]]\\ \mathsf{t}:\emptyset & \text{otherwise}\end{cases}$$

That is, (i) if T is private type $\mathsf{t}[\mathsf{g}]$ then Δ_T^U is the permission $\mathsf{t}:\mathsf{update}$ since the process is writing an object of type $\mathsf{t}[\mathsf{g}]$, (ii) if $T=\mathsf{G}[\mathsf{t}[\mathsf{g}]]$ then Δ_T^U is the permission $\mathsf{disseminate\ G\ 1}$ since the process is disclosing once a link to private data via a channel of group G, and (iii) the empty set of permissions otherwise.

Rule [TInp] is similar, except that the permission interface generated contains the permissions exercised by process P by the input value k along with permissions Δ_T^R, where set Δ_T^R is defined by:

$$\Delta_T^R \ = \ \begin{cases}\mathsf{t}:\{\mathsf{read}\} & \text{if} \quad T=\mathsf{t}[\mathsf{g}]\\ \mathsf{t}:\{\mathsf{reference}\} & \text{if} \quad T=\mathsf{G}[\mathsf{t}[\mathsf{g}]]\\ \mathsf{t}:\emptyset & \text{otherwise}\end{cases}$$

That is Δ_T^R contains (i) permission read if T is private type $\mathsf{t}[\mathsf{g}]$ since the process is reading an object of type $\mathsf{t}[\mathsf{g}]$, (ii) permission $\mathsf{reference}$ if $T=\mathsf{G}[\mathsf{t}[\mathsf{g}]]$ since in this case the process is reading a reference to a private object and (iii) the empty set otherwise.

For name restriction, [TRes] specifies that if P type checks within an environment $\Gamma;\Lambda$, then $(\nu n)P$ type checks in an environment $\Gamma\backslash\{n\};\Lambda\backslash\{n\};Z$, i.e. without name n. Moving on to rule [TRepl] we have that if a process P produces an interface Δ then $*P$ produces an

interface $\Delta^* \uplus \Delta'$, where

$$\begin{array}{rcl} \Delta^* & = & \{\mathsf{prm} \mid \mathsf{prm} \in \Delta, \mathsf{prm} \neq \mathsf{disseminate\ G}\ \lambda\} \\ & \cup & \{\mathsf{disseminate\ G}\ \omega \mid \mathsf{disseminate\ G}\ \lambda \in \Delta\} \end{array}$$

In words, (i) Δ^* is such that if a type is disclosed $\lambda > 0$ times in Δ then it is disclosed for an unlimited number of times in Δ^* and (ii) Δ' contains the permission aggregate for all stores of type t in P. This is because the store replication may allow a system component to store multiple private data for the same individual. In turn, rules [TPar] uses the $\uplus$ operator to compose the process interfaces of P_1 and P_2 while additionally including any aggregation being exercised by the two systems. Such aggregation takes place if the parallel components have the capacity of storing more than one piece of information for the same individual. This is the case either if they possess more than one store for the same identifier, or if they have more than one store and one of the stores has not yet been instantiated (i.e. the identity field is a variable), thus, has the capacity of user-based data aggregation.

Finally, rule [TIf] extracts the permission interface of a conditional process as the collection of the permissions exercised by each of the component processes as well as those exercised by the condition. Note that in this and the previous rule, Λ_1 and Λ_2 are assumed to be disjoint, and Z_1, Z_2 represent the concatenation of the two environments.

5.4. **A type system for system terms.** In order to produce a permission interface for systems, we employ the following typing environment which extends the typing environment Δ with information regarding the groups being active while executing permissions for different types of private data. Specifically, we write:

$$\Theta \quad ::= \quad \Theta, \mathsf{t} : \theta \mid \emptyset$$

Thus, θ is a flat hierarchy of the form $\mathsf{G}_1[\mathsf{G}_2[\ldots \mathsf{G}_n[\tilde{\mathsf{prm}}]\ldots]]$ associating a sequence of groups with a set of permissions, and Θ is a collection of such associations for different private types. Structure θ is called interface hierarchy. The typing judgement for systems becomes:

$$\Gamma; \Lambda \vdash S \triangleright \Theta$$

and captures that system S is well-typed in type environment Γ and linear store environment Λ and produces the interface Θ which records the group memberships of all components of S as well as the permissions exercised by each of the components. Figure 7 presents the associated typing rules. The first rule employed at the system level is rule [TGr] according to which, if P produces a typing Δ, then system $\mathsf{G}[P]$ produces the Θ-interface where group G is applied to all components of Δ. This captures that there exists a component in the system belonging to group G and exercising permissions as defined by Δ. For rule [TSGr], we have that if S produces a typing interface Θ then process $\mathsf{G}[S]$ produces a new interface where all components of Θ are extended by adding group G to the group membership of all components. Next, for the parallel composition of systems, rule [TSPar], concatenates the system interfaces of S_1 and S_2, thus collecting the permissions exercised by each of components of both of S_1 and S_2. Finally, for name restriction, [TSRes] specifies that if S type checks within an environment $\Gamma; \Lambda$, then $(\nu n)S$ type checks in environment $\Gamma \backslash \{n\}; \Lambda \backslash \{n\}$.

Example 5.1. As an example we type the Lab system from Section 2.3:

$$\mathsf{Lab}[b?(w).\, w?(x \otimes y).\, r?(\_ \otimes z).\, \texttt{if}\ y = z\ \texttt{then}\ c!\langle w \rangle.\, \mathbf{0}\ \texttt{else}\ \mathbf{0}]$$

$$[\mathsf{TGr}]\ \frac{\Gamma;\Lambda;Z\vdash P\triangleright\Delta \quad \Theta=\{\mathsf{t}:\mathsf{G}[\tilde{\mathsf{prm}}]\mid \mathsf{t}:\tilde{\mathsf{prm}}\in\Delta\}}{\Gamma;\Lambda\vdash \mathsf{G}[P]\triangleright\Theta} \qquad [\mathsf{TSGr}]\ \frac{\Gamma;\Lambda\vdash S\triangleright\Theta \quad \Theta'=\{\mathsf{t}:\mathsf{G}[\theta]\mid \mathsf{t}:\theta\in\Theta\}}{\Gamma;\Lambda\vdash \mathsf{G}[S]\triangleright\Theta'}$$

$$[\mathsf{TSPar}]\ \frac{\Gamma;\Lambda_i\vdash S_i\triangleright\Theta_i \quad i\in\{1,2\}}{\Gamma;\Lambda_1,\Lambda_2\vdash S_1\|S_2\triangleright\Theta_1,\Theta_2} \qquad [\mathsf{TSRes}]\ \frac{\Gamma;\Lambda\vdash S\triangleright\Theta}{\Gamma\backslash\{n\};\Lambda\backslash\{n\}\vdash(\nu\ n)S\triangleright\Theta}$$

Figure 7: Typing rules for systems

Consider also a type environment:

$$\begin{aligned}\Gamma \;=\;& r:\mathsf{Police}[\mathsf{crime}[\mathsf{dna}]],\_\otimes z:\mathsf{crime}[\mathsf{dna}],\\ & w:\mathsf{Hospital}[\mathsf{patient\_data}[\mathsf{dna}]],x\otimes y:\mathsf{patient\_data}[\mathsf{dna}]\\ & b:\mathsf{Hospital}[\mathsf{Hospital}[\mathsf{crime}[\mathsf{dna}]]],c:\mathsf{Police}[\mathsf{Hospital}[\mathsf{crime}[\mathsf{dna}]]]\end{aligned}$$

In system Lab, name r and variable w have a reference type and are used to access private data of type crime and patient_data, respectively. Name r will be used to receive data from the police database. Placeholders $\_\otimes z$ and $x\otimes y$ instantiate private data of type crime and patient_data, respectively. Name b is used to substitute reference variable w and name c is used to send reference w to the group Police.

To type the matching expression we use typing rule [TId]:

$$[\mathsf{TId}]\ \frac{\Gamma\vdash x\otimes y:\mathsf{patient\_data}[\mathsf{dna}]\triangleright\mathsf{patient\_data}:\{\mathsf{readId}\} \quad \Gamma\vdash\_\otimes z:\mathsf{crime}[\mathsf{dna}]\triangleright\mathsf{crime}:\emptyset}{\Gamma\vdash y=z\triangleright\mathsf{crime}:\{\mathsf{identify}\{\mathsf{patient\_data}\}\},\mathsf{patient\_data}:\{\mathsf{readId}\}}$$

In the matching expression above we have private data placeholder $\_\otimes z$ to be of type crime[dna] (private data that is anonymised) and placeholder $x\otimes y$ of type patient_data[dna] with x being known (permission readId). Both variables z and y represent the dna type and can be matched against each other. The fact that z comes from an anonymous private data and y data has a known data subject implies the process of identification of crime data against patient_data, which is expressed as mapping crime: {identify{patient_data}} in the permission typing.

The conditional term is typed using typing rule [TIf]:

$$[\mathsf{TIf}]\ \frac{\Gamma\vdash y=z\triangleright\mathsf{crime}:\{\mathsf{identify}\{\mathsf{patient\_data}\}\},\mathsf{patient\_data}:\{\mathsf{readId}\} \quad \Gamma;\emptyset;\emptyset\vdash c!\langle w\rangle.\,\mathbf{0}\triangleright\mathsf{patient\_data}:\{\mathsf{disseminate}\ \mathsf{Police}\ 1\}}{\Gamma;\emptyset;\emptyset\vdash \mathtt{if}\ y=z\ \mathtt{then}\ c!\langle w\rangle.\,\mathbf{0}\ \mathtt{else}\ \mathbf{0}\triangleright\Delta}$$

with $\Delta=\mathsf{crime}:\{\mathsf{identify}\{\mathsf{patient\_data}\}\},\mathsf{patient\_data}:\{\mathsf{readId},\mathsf{disseminate}\ \mathsf{Police}\ 1\}$. The conditional term types the two branches of the term and identifies the kind of data processing in the matching operator. The branch $b!\langle w\rangle.\,\mathbf{0}$ results in the permission environment patient_data: {disseminate Police 1} because reference w is being disseminated. The resulting permission environment is the $\uplus$-union of the three former permission environments.

The whole process of the system is typed as:

$$\frac{\Gamma;\emptyset;\emptyset\vdash w?(x\otimes y).\,r?(\_\otimes z).\,\mathtt{if}\ y=z\ \mathtt{then}\ c!\langle w\rangle.\,\mathbf{0}\ \mathtt{else}\ \mathbf{0}\triangleright\Delta_1}{\Gamma;\emptyset;\emptyset\vdash b?(w).\,w?(x\otimes y).\,r?(\_\otimes z).\,\mathtt{if}\ y=z\ \mathtt{then}\ c!\langle w\rangle.\,\mathbf{0}\ \mathtt{else}\ \mathbf{0}\triangleright\Delta_1\uplus\Delta_2}$$

with

$$\Delta_1 = \mathsf{patient\_data}{:}\{\mathsf{read}, \mathsf{readId}, \mathsf{disseminate\ Police\ 1}\}, \mathsf{crime}{:}\{\mathsf{read}, \mathsf{identify}\{\mathsf{patient\_data}\}\}$$
$$\Delta_2 = \mathsf{patient\_data}{:}\{\mathsf{reference}\}$$

Finally, we use typing rule [TSGr] to type the whole system:

$$[\mathsf{TSGr}]\ \frac{\Gamma; \emptyset; \emptyset \vdash b?(w).\, w?(x \otimes y).\, r?(\_ \otimes z).\, \mathtt{if}\ y = z\ \mathtt{then}\ c!\langle w\rangle.\, \mathbf{0}\ \mathtt{else}\ \mathbf{0} \triangleright \Delta_1 \uplus \Delta_2}{\Gamma; \emptyset \vdash \mathsf{Lab}[b?(w).\, w?(x \otimes y).\, r?(\_ \otimes z).\, \mathtt{if}\ y = z\ \mathtt{then}\ c!\langle w\rangle.\, \mathbf{0}\ \mathtt{else}\ \mathbf{0}] \triangleright \Theta}$$

with

$$\Theta = \begin{array}{l} \mathsf{patient\_data} : \mathsf{Lab}[\mathsf{reference}, \mathsf{read}, \mathsf{readId}, \mathsf{disseminate\ Police\ 1}], \\ \mathsf{crime} : \mathsf{Lab}[\mathsf{read}, \mathsf{identify}\{\mathsf{patient\_data}\}] \end{array}$$

Rule [TSGr] constructs a permission interface Θ using the typing environment from the containing process. In this case the rule constructs two interface hierarchies on group hierarchy Lab using the permissions for patient_data and crime in the process's typing environment.

6. Soundness and Safety

In this section we establish soundness and safety results for our framework. The first result we establish is the result of type preservation where we show that the type system is preserved by the labelled transition system. Type preservation is then used to prove a safety property of a system with respect to privacy policies. A basic notion we introduce in this section is the notion of policy satisfaction which we show to be preserved by the semantics. Finally, we show that a safe system would never reduce to an error system through a safety theorem.

6.1. Type Preservation.

We first establish standard weakening and strengthening lemmas.

Lemma 6.1 (Weakening)**.**

- *Let P be a process with $\Gamma; \Lambda; Z \vdash P \triangleright \Delta$ and $v \notin \mathsf{fn}(P)$. Then $\Gamma, v : T; \Lambda; Z \vdash P \triangleright \Delta$.*
- *Let P be a process with $\Gamma; \Lambda; Z \vdash P \triangleright \Delta$ and $\mathsf{fv}(k) \cap \mathsf{fv}(P) = \emptyset$. Then $\Gamma, k : T; \Lambda; Z \vdash P \triangleright \Delta$.*
- *Let S be a system with $\Gamma; \Lambda \vdash S \triangleright \Theta$ and $v \notin \mathsf{fn}(S)$. Then $\Gamma, v : T; \Lambda \vdash S \triangleright \Theta$.*
- *Let S be a system with $\Gamma; \Lambda \vdash S \triangleright \Theta$ and $\mathsf{fv}(k) \cap \mathsf{fv}(S) = \emptyset$. Then $\Gamma, k : T; \Lambda \vdash S \triangleright \Theta$.*

Proof. The proof for Part 1 and 2 (resp., Part 3 and 4) is a standard induction on the typing derivation of P (resp., S). □

Lemma 6.2 (Strengthening)**.**

- *Let P be a process with $\Gamma, v : T; \Lambda; Z \vdash P \triangleright \Delta$ and $v \notin \mathsf{fn}(P)$. Then $\Gamma; \Lambda; Z \vdash P \triangleright \Delta$.*
- *Let P be a process with $\Gamma, k : T; \Lambda; Z \vdash P \triangleright \Delta$ and $\mathsf{fv}(k) \cap \mathsf{fv}(P) = \emptyset$. Then $\Gamma; \Lambda; Z \vdash P \triangleright \Delta$.*
- *Let S be a system with $\Gamma, v : T; \Lambda \vdash S \triangleright \Theta$ and $v \notin \mathsf{fn}(S)$. Then $\Gamma; \Lambda \vdash S \triangleright \Theta$.*
- *Let S be a system with $\Gamma, k : T; \Lambda \vdash S \triangleright \Theta$ and $\mathsf{fv}(k) \cap \mathsf{fv}(P) = \emptyset$. Then $\Gamma; \Lambda \vdash S \triangleright \Theta$.*

Proof. The proof for Part 1 and 2 (resp., Part 3 and 4) is a standard induction on the typing derivation of P (resp., S). □

We may now establish that typing is preserved under substitution.

Lemma 6.3 (Substitution)**.** *Let* $v_2 \notin \mathsf{domain}(\Gamma)$.

- *If* $\Gamma, v_1 : T; \Lambda; Z \vdash P \triangleright \Delta$ *then one of the following holds:*
  - $\Gamma, v_2 : T; \Lambda; Z \vdash P\{^{v_2}/_{v_1}\} \triangleright \Delta$
  - $\Gamma, v_2 : T; (\Lambda \backslash v_1), v_2; Z \vdash P\{^{v_2}/_{v_1}\} \triangleright \Delta$
  - $\Gamma, v_2 : T; \Lambda; (Z \backslash \langle \iota, \mathsf{t} \rangle), \langle \mathsf{id}, \mathsf{t} \rangle \vdash P\{^{v_2}/_{v_1}\} \triangleright \Delta$, *if* $v_1 = \iota \otimes \delta \wedge v_2 = \mathsf{id} \otimes c \wedge T = \mathsf{t}[\mathsf{g}]$
- *If* $\Gamma, k : T; \Lambda; Z \vdash P \triangleright \Delta$ *then one of the following holds:*
  - $\Gamma, v_2 : T; \Lambda; Z \vdash P\{^{v_2}/_{k}\} \triangleright \Delta$
  - $\Gamma, v_2 : T; \Lambda; (Z \backslash \langle x, \mathsf{t} \rangle), \langle \mathsf{id}, \mathsf{t} \rangle \vdash P\{^{v_2}/_{v_1}\} \triangleright \Delta$, *if* $k = x \otimes y \wedge v_2 = \mathsf{id} \otimes c \wedge T = \mathsf{t}[\mathsf{g}]$

Proof. The proof is an induction on the structure of the syntax of P. We give the interesting case of substituting a value inside a store.

- Let $P = \overline{r} \triangleright [\mathsf{id} \otimes c]$ and $\Gamma, \mathsf{id} \otimes c : \mathsf{t}[\mathsf{g}]; r; \mathsf{id} : \mathsf{t} \vdash P \triangleright \Delta, \mathsf{t} : \{\mathsf{store}\}$. By the premise of typing rule [TSt] used to type the process P, we get:
$$\Gamma, \mathsf{id} \otimes c : \mathsf{t}[\mathsf{g}] \vdash \mathsf{id} \otimes c : \mathsf{t}[\mathsf{g}] \triangleright \Delta'$$
By applying strengthening (Lemma 6.2) to $\Gamma, \mathsf{id} \otimes c : \mathsf{t}[\mathsf{g}]$ and then weakening (Lemma 6.1) we get:
$$\Gamma, \mathsf{id} \otimes c' : \mathsf{t}[\mathsf{g}] \vdash \mathsf{id} \otimes c' : \mathsf{t}[\mathsf{g}] \triangleright \Delta' \tag{6.1}$$
Assume now that:
$$P\{^{\mathsf{id} \otimes c'}/_{\mathsf{id} \otimes c}\} = \overline{r} \triangleright [\mathsf{id} \otimes c']$$
If we use result 6.1 we can apply the typing rule [TSt] to the latter substitution and obtain:
$$\Gamma, \mathsf{id} \otimes c' : \mathsf{t}[\mathsf{g}]; u; \mathsf{id} : \mathsf{t} \vdash P\{^{\mathsf{id} \otimes c'}/_{\mathsf{id} \otimes c}\} \triangleright \Delta, \mathsf{t} : \{\mathsf{store}\}$$
as required.
- Let $P = \overline{r} \triangleright [x \otimes y]$ and $\Gamma, x \otimes y : \mathsf{t}[\mathsf{g}]; r; x : \mathsf{t} \vdash P \triangleright \Delta, \mathsf{t} : \{\mathsf{store}\}$. By the premise of the typing rule [TSt] used to type the process P, we get:
$$\Gamma, x \otimes y : \mathsf{t}[\mathsf{g}] \vdash x \otimes y : \mathsf{t}[\mathsf{g}] \triangleright \Delta'$$
By applying strengthening (Lemma 6.2) to $\Gamma, x \otimes y : \mathsf{t}[\mathsf{g}]$ and then weakening (Lemma 6.1) we obtain:
$$\Gamma, \mathsf{id} \otimes c : \mathsf{t}[\mathsf{g}] \vdash \mathsf{id} \otimes c : \mathsf{t}[\mathsf{g}] \triangleright \Delta' \tag{6.2}$$
Assume now that:
$$P\{^{\mathsf{id} \otimes c}/_{x \otimes y}\} = \overline{r} \triangleright [\mathsf{id} \otimes c]$$
If we use result 6.2 we can apply the typing rule [TSt] to the latter substitution and get:
$$\Gamma, \mathsf{id} \otimes c : \mathsf{t}[\mathsf{g}]; r; \mathsf{id} : \mathsf{t} \vdash P\{^{\mathsf{id} \otimes c}/_{x \otimes y}\} \triangleright \Delta, \mathsf{t} : \{\mathsf{store}\}$$
as required.
- Let $P = \overline{r} \triangleright [\iota \otimes \delta]$ and $\Gamma, r : \mathsf{G}[\mathsf{t}[\mathsf{g}]]; u; \iota : \mathsf{t} \vdash P \triangleright \Delta, \mathsf{t} : \{\mathsf{store}\}$. By the premise of the typing rule [TSt] used to type the process P, we get:
$$\Gamma, r : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \vdash r : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \triangleright \emptyset$$
By applying strengthening (Lemma 6.2) to $\Gamma, r : \mathsf{G}[\mathsf{t}[\mathsf{g}]]$ and then weakening (Lemma 6.1) we obtain:
$$\Gamma, r' : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \vdash u' : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \triangleright \emptyset \tag{6.3}$$

Assume now that:

$$P\{^{r'}/_{r}\} = \overline{r'} \triangleright [\iota \otimes \delta]$$

If we use result 6.3 we can apply the typing rule [TSt] to the latter substitution and get:

$$\Gamma, r' : \mathsf{G}[\mathsf{t}[\mathsf{g}]]; r'; \iota : \mathsf{t} \vdash P\{^{r'}/_{r}\} \triangleright \Delta, \mathsf{t} : \{\mathsf{store}\}$$

as required. □

We next define a relation $\preceq$ over permissions and the structures of permissions to capture the changes in the permission environment and the interface environment during action execution.

Definition 6.4 ($\Theta_1 \preceq \Theta_2$)**.** We define the relation $\preceq$ over permissions (prm), permissions environments (Δ), and interface environments (Θ) as follows:

(1) $\tilde{\mathsf{prm}}_1 \preceq \tilde{\mathsf{prm}}_2$, whenever
  (i) for all $\mathsf{prm} \in \tilde{\mathsf{prm}}_1$ such that $\mathsf{prm} \neq \mathsf{disseminate\ G}\ \lambda$ and $\mathsf{prm} \neq \mathsf{usage}\{\tilde{\mathsf{p}}\}$ we have that $\mathsf{prm} \in \tilde{\mathsf{prm}}_2$.
  (ii) for all $\mathsf{prm} \in \tilde{\mathsf{prm}}_1$ such that $\mathsf{prm} = \mathsf{disseminate\ G}\ \lambda_1$ we have that $\mathsf{disseminate\ G}\ \lambda_2 \in \tilde{\mathsf{prm}}_2$ and $(\lambda_1 \leq \lambda_2 \vee \lambda_2 = *)$.
  (iii) for all $\mathsf{prm} \in \tilde{\mathsf{prm}}_1$ such that $\mathsf{prm} = \mathsf{usage}\{\tilde{\mathsf{p}}_1\}$ we have that $\mathsf{usage}\{\tilde{\mathsf{p}}_2\} \in \tilde{\mathsf{prm}}_2$ and $\tilde{\mathsf{p}}_1 \subseteq \tilde{\mathsf{p}}_2$.
(2) $\Delta_1 \preceq \Delta_2$, whenever for all t such that $\mathsf{t} : \tilde{\mathsf{prm}}_1 \in \Delta_1$ we have that $\mathsf{t} : \tilde{\mathsf{prm}}_2 \in \Delta_2$ and $\tilde{\mathsf{prm}}_1 \preceq \tilde{\mathsf{prm}}_2$.
(3) $\mathsf{G}[\tilde{\mathsf{prm}}_1] \preceq \mathsf{G}[\tilde{\mathsf{prm}}_2]$ whenever $\tilde{\mathsf{prm}}_1 \preceq \tilde{\mathsf{prm}}_2$.
(4) $\mathsf{G}[\theta_1] \preceq \mathsf{G}[\theta_2]$ whenever $\theta_1 \preceq \theta_2$.
(5) $\Theta_1 \preceq \Theta_2$, whenever for all t such that $\mathsf{t} : \theta_1 \in \Theta_1$ we have that $\mathsf{t} : \theta_2 \in \Theta_2$ and $\theta_1 \preceq \theta_2$.

The following proposition follows directly from the definition above.

Proposition 6.5.

- *Let $i \in \{1, 2\}$. Then $\Delta_i \preceq \Delta_1 \uplus \Delta_2$.*
- *Let $\Delta_1 \preceq \Delta_2$. Then $\Delta_1 \uplus \Delta \preceq \Delta_2 \uplus \Delta$.*

Proof. The proof is immediate from the definition of $\uplus$ and $\preceq$. □

We may now state type preservation by action execution of a system. Type preservation employs the $\preceq$ operator. Specifically, when a process or a system executes an action we expect a name to maintain or lose its interface capabilities.

Theorem 6.6 (Type Preservation)**.** *Consider a process P and a system S.*

- *If $\Gamma; \Lambda; Z \vdash P \triangleright \Delta$ and $P \xrightarrow{\ell} P'$ then*
  – *if $\ell \neq n?(v)$ then $\Gamma; \Lambda; Z \vdash P' \triangleright \Delta'$ and $\Delta' \preceq \Delta$.*
  – *if $\ell = n?(v)$ then for some Λ' and Z', we have $\Gamma, v : T; \Lambda; Z \vdash P' \triangleright \Delta'$ and $\Delta' \preceq \Delta$.*
- *If $\Gamma; \Lambda \vdash S \triangleright \Theta$ and $S \xrightarrow{\ell} S'$*
  – *if $\ell \neq n?(v)$ then $\Gamma; \Lambda \vdash S' \triangleright \Theta'$ and $\Theta' \preceq \Theta$.*
  – *if $\ell = n?(v)$ then for some Λ', we have $\Gamma, v : T; \Lambda' \vdash S' \triangleright \Theta'$ and $\Theta' \preceq \Theta$.*

Proof. The proof is by induction on the inference tree for $\xrightarrow{\ell}$. We begin with the proof of the first part.

Base Step:

- Case: $n!\langle v\rangle.\, P \xrightarrow{n!\langle v\rangle} P$ and $\Gamma;\Lambda;Z \vdash n!\langle v\rangle.\, P \triangleright \Delta$. By the premise of typing rule [TOut] we get that $\Gamma;\Lambda;Z \vdash P \triangleright \Delta'$ with $\Gamma \vdash v \triangleright \Delta'$ and $\Delta = \Delta' \uplus \Delta^U_T$. The result is then immediate from Proposition 6.5.
- Case: $n?(k).\, P \xrightarrow{n?(v)} P\{^v/_k\}$ and $\Gamma;\Lambda;Z \vdash n?(k).\, P \triangleright \Delta$. By the premise of typing rule [TInp] we get that $\Gamma;\Lambda;Z \vdash P \triangleright \Delta_1$ and $\Gamma \vdash v \triangleright \Delta_2$ with $\Delta = \Delta_1 \uplus \Delta_2 \uplus \Delta^R_T$. From weakening (Lemma 6.1) and the Substitution Lemma 6.3 we get that for some Λ' and Z', $\Gamma, v : T;\Lambda';Z' \vdash P\{^v/_k\} \triangleright \Delta_1$. The result is then immediate from Proposition 6.5.
- Case: $\overline{r} \triangleright [\mathsf{id} \otimes c] \xrightarrow{\overline{r}!\langle c\rangle} \overline{r} \triangleright [\mathsf{id} \otimes c]$. The case is trivial.
- Case: $\overline{r} \triangleright [\mathsf{id} \otimes c] \xrightarrow{\overline{r}?(c')} \overline{r} \triangleright [\mathsf{id} \otimes c']$ and $\Gamma; r; \mathsf{id} : \mathsf{t} \vdash \overline{r} \triangleright [\mathsf{id} \otimes c] \triangleright \Delta$. The result is immediate by the Substitution Lemma 6.3.

  Induction Step:
- The interesting case for the induction step follows the structure of the τ interaction (LTS rule [Tau]). $P \mid Q \xrightarrow{\tau} (\nu\ \tilde{m})(P' \mid Q')$ and $\Gamma;\Lambda_P,\Lambda_Q;Z_P,Z_Q \vdash P \mid Q \triangleright \Delta_P \uplus \Delta_Q \uplus \Delta$. By the premise of LTS rule [Tau] and typing rule [TPar] we get:

$$\begin{array}{rcl} P & \xrightarrow{\ell_1} & P' \text{ and } \Gamma;\Lambda_P;Z_P \vdash P \triangleright \Delta_P \\ Q & \xrightarrow{\ell_2} & Q' \text{ and } \Gamma;\Lambda_Q;Z_Q \vdash Q \triangleright \Delta_Q \\ \tilde{m} & = & \mathsf{bn}(\ell_1) \cup \mathsf{bn}(\ell_2) \end{array}$$

  By the induction hypothesis we know that

$$\Gamma;\Lambda_P;Z_P \vdash P' \triangleright \Delta'_P \text{ and } \Delta'_P \preceq \Delta_P$$
$$\Gamma;\Lambda_Q;Z_Q \vdash Q' \triangleright \Delta'_Q \text{ and } \Delta'_Q \preceq \Delta_Q$$

  We apply type rule [TPar] together with type rule [TRes] to get:

$$\Gamma\backslash\tilde{m};(\Lambda_P,\Lambda_Q)\backslash\tilde{m};Z_P,Z_Q \vdash (\nu\ \tilde{m})(P' \mid Q') \triangleright \Delta'_P \uplus \Delta'_Q \uplus \Delta$$

  The result is then immediate from Proposition 6.5.
- The rest of the induction step cases follow easier argumentations.

We continue with the proof of the second part of the Theorem. The interesting case is the case where $S = \mathsf{G}[P]$. The rest of the cases are congruence cases that follow argumentations similar to those in the previous part.

- Case: $\mathsf{G}[P] \xrightarrow{\ell} \mathsf{G}[P']$ with $\Gamma;\Lambda \vdash G[P] \triangleright \Theta$. By the premise of LTS rule [SPGr] and typing rule [TGr] we get:

$$\begin{array}{c} P \quad \xrightarrow{\ell} \quad P' \\ \Gamma;\Lambda;Z \vdash P \triangleright \Delta \quad \text{with} \quad \forall \mathsf{t} : \mathsf{p}\tilde{\mathsf{r}}\mathsf{m} \in \Delta \iff \mathsf{t} : G[\mathsf{p}\tilde{\mathsf{r}}\mathsf{m}] \in \Theta \end{array} \tag{6.4}$$

  Part 1 of this theorem ensures that for some $\Gamma' = \emptyset$ if $\ell \neq n?(v)$ and $\Gamma' = v : T$ if $\ell = n?(v)$ and some Λ' and Z', $\Gamma,\Gamma';\Lambda';Z' \vdash P' \triangleright \Delta'$ with $\Delta' \preceq \Delta$. If we apply typing rule [TGr] we get:

$$\Gamma,\Gamma';\Lambda' \vdash G[P'] \triangleright \Theta' \quad \text{with} \quad \forall \mathsf{t} : \mathsf{p}\tilde{\mathsf{r}}\mathsf{m} \in \Delta' \iff \mathsf{t} : G[\mathsf{p}\tilde{\mathsf{r}}\mathsf{m}] \in \Theta'$$

  We can derive that $\Theta' \preceq \Theta$ from equation 6.4 and the fact that $\Delta' \preceq \Delta$.
- The rest of the cases are similar. □

6.2. **Policy Satisfaction.** We define the notion of *policy satisfaction*. Policy satisfaction uses the type system as the bridge to establish a relation between a system and a privacy policy. Intuitively a system satisfies a policy if its interface environment satisfies a policy.

Working towards policy satisfaction, we first define a satisfaction relation between policies $\mathcal{P}$ and permission interfaces Θ.

Definition 6.7. We define two satisfaction relations, denoted $\Vdash$, as:

- Consider a policy hierarchy H and an interface hierarchy θ. We say that θ *satisfies* H, written $\mathsf{H} \Vdash \theta$, whenever:

$$\frac{\exists k \in J : \mathsf{H}_k = \mathsf{G}' : \tilde{\mathsf{prm}'}[\mathsf{H}_i]_{i\in I} \quad \mathsf{G}' : \tilde{\mathsf{prm}'} \uplus \tilde{\mathsf{prm}}[\mathsf{H}_i]_{i\in I} \Vdash \theta}{\mathsf{G} : \tilde{\mathsf{prm}}[\mathsf{H}_j]_{j\in J} \Vdash \mathsf{G}[\theta]} \qquad \frac{\tilde{\mathsf{prm}}_2 \preceq \tilde{\mathsf{prm}}_1}{\mathsf{G} : \tilde{\mathsf{prm}}_1[] \Vdash \mathsf{G}[\tilde{\mathsf{prm}}_2]}$$

- Consider a policy $\mathcal{P}$ and an interface Θ. Θ *satisfies* $\mathcal{P}$, written $\mathcal{P} \Vdash \Theta$, whenever:

$$\frac{\mathsf{H} \Vdash \theta}{\mathsf{t} \gg \mathsf{H}; \mathcal{P} \Vdash \mathsf{t} : \theta} \qquad \frac{\mathsf{H} \Vdash \theta \qquad \mathcal{P} \Vdash \Theta}{\mathsf{t} \gg \mathsf{H}; \mathcal{P} \Vdash \mathsf{t} : \theta; \Theta}$$

According to the definition of $\mathsf{H} \Vdash \theta$, an interface hierarchy θ satisfies a policy hierarchy H whenever: i) the θ group hierarchy is included in group hierarchy H; and ii) the permissions of the interface hierarchy are included in the union of the permissions of the corresponding group hierarchy in H. Similarly, a Θ-interface satisfies a policy, $\mathcal{P} \Vdash \Theta$, if for each component $\mathsf{t} : \theta$ of Θ, there exists a component $\mathsf{t} \gg \mathsf{H}$ of $\mathcal{P}$ such that θ satisfies H. At this point, note that we assume ; to be a commutative operator. A direct corollary of the definition is the preservation of the $\preceq$ operator over the satisfiability relation:

Proposition 6.8. *If* $\mathcal{P} \Vdash \Theta_1$ *and* $\Theta_2 \preceq \Theta_1$ *then* $\mathcal{P} \Vdash \Theta_2$.

Policy satisfaction is a main definition for this paper as it specifies the situation where a system respects a privacy policy. The formalisation of the intuition above follows in the next definition.

Definition 6.9 (Policy Satisfaction)**.** Consider $\mathcal{P} : \diamond$, a type environment Γ and system S. We say that S satisfies $\mathcal{P}$, written $\mathcal{P}; \Gamma \vdash S$, whenever there exist Λ and Θ such that $\Gamma; \Lambda \vdash S \triangleright \Theta$ and $\mathcal{P} \Vdash \Theta$.

The main idea is to check whether a system satisfies a given privacy policy under an environment that maps channels, constants, and private data to channel types, constant types, and private data types, respectively.

Corollary 6.10. *If* $\mathcal{P}; \Gamma \vdash S$ *and* $S \xrightarrow{\ell} S'$ *then* $\mathcal{P}; \Gamma \vdash S'$

Proof. The proof is direct from Corollary 6.8 and Theorem 6.6. □

6.3. **System Safety.** We consider a system to be safe with respect to a privacy policy when the system cannot reduce to an error system, i.e., a state where the privacy policy is violated. Towards this direction we need to define a class of error systems that is parametrised on privacy policies.

We begin with an auxiliary definition of $\mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$ that counts the number of output prefixes of the form $u!\langle t \rangle.\, Q$, where $t : \mathsf{t}[\mathsf{g}] \in \Gamma$, within a process P. The induction definition of function $\mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$ is as follows.

Definition 6.11 (Count References)**.** We define function $\mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$ as:

- $\mathsf{countRef}(\mathbf{0}, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) = 0$
- $\mathsf{countRef}(u!\langle t\rangle.\, P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) = 1 + \mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$ if $t : \mathsf{t}[\mathsf{g}] \in \Gamma$
- $\mathsf{countRef}(u!\langle t\rangle.\, P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) = \mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$ if $t : \mathsf{t}[\mathsf{g}] \notin \Gamma$
- $\mathsf{countRef}(\overline{u} \triangleright [\iota \otimes \delta], \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) = 0$
- $\mathsf{countRef}(u?(k).\, P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) = \mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$
- $\mathsf{countRef}((\nu\ n)P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) = \mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$
- $\mathsf{countRef}((\nu\ n)P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) = \mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$
- $\mathsf{countRef}(P \mid Q, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) = \mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) + \mathsf{countRef}(Q, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$
- $\mathsf{countRef}(\texttt{if}\ u = u'\ \texttt{then}\ P\ \texttt{else}\ Q, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$
  $= \mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) + \mathsf{countRef}(Q, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]])$

We may now define the notion of an *error system* with respect to a privacy policy. Intuitively an error system is a system that can do an action not permitted by the privacy policy. The error system clarifies the satisfiability relation between policies and processes.

Definition 6.12 (Error System)**.** Assume $\tilde{\mathsf{G}} = \mathsf{G}_1, \ldots, \mathsf{G}_n$, and consider a policy $\mathcal{P}$, an environment Γ, and a system

$$\mathsf{System} \equiv \mathsf{G}_1[(\nu\ \tilde{x}_1)(\mathsf{G}_2[(\nu\ \tilde{x}_2)(\ldots(\mathsf{G}_n[(\nu\ \tilde{x}_n)(P \mid Q)] \| S_n)\ldots)])] \| S_1$$

System System is an *error system* with respect to $\mathcal{P}$ and Γ if there exists t such that $\mathcal{P} = \mathsf{t} \gg \mathsf{H}; \mathcal{P}'$ and at least one of the following holds:

(1) $\mathsf{read} \notin \mathsf{perms}(\mathsf{H}_{\tilde{\mathsf{G}}})$ and $\exists u$ such that $\Gamma \vdash u : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \triangleright \Delta$ and $P \equiv u?(k).\, P'$.
(2) $\mathsf{update} \notin \mathsf{perms}(\mathsf{H}_{\tilde{\mathsf{G}}})$ and $\exists u$ such that $\Gamma \vdash u : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \triangleright \Delta$ and $P \equiv u!\langle v\rangle.\, P'$.
(3) $\mathsf{reference} \notin \mathsf{perms}(\mathsf{H}_{\tilde{\mathsf{G}}})$ and $\exists k$ such that $\Gamma \vdash k : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \triangleright \emptyset$ and $P \equiv u?(k).\, P'$.
(4) $\mathsf{disseminate}\ \mathsf{G}'\ \lambda \notin \mathsf{perms}(\mathsf{H}_{\tilde{\mathsf{G}}})$ and $\exists u$ such that $\Gamma \vdash u : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \triangleright \emptyset$ and $P \equiv u'!\langle u\rangle.\, P'$.
(5) $\mathsf{readId} \notin \mathsf{perms}(\mathsf{H}_{\tilde{\mathsf{G}}})$ and $\exists u$ such that $\Gamma \vdash u : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \triangleright \Delta$ and $P \equiv u?(x \otimes y).\, P'$.
(6) $\mathsf{store} \notin \mathsf{perms}(\mathsf{H}_{\tilde{\mathsf{G}}})$ and $\exists r$ such that $\Gamma \vdash r : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \triangleright \Delta$ and $P \equiv \overline{r} \triangleright [\mathsf{id} \otimes \delta]$.
(7) $\mathsf{aggregate} \notin \mathsf{perms}(\mathsf{H}_{\tilde{\mathsf{G}}})$ and $\exists u$ such that $\Gamma \vdash r : \mathsf{G}[\mathsf{t}[\mathsf{g}]] \triangleright \Delta$ and $P \equiv \overline{r} \triangleright [\mathsf{id} \otimes \delta] \mid \overline{r'} \triangleright [\mathsf{id} \otimes \delta']$.
(8) $\mathsf{usage}\{\mathsf{p}\} \notin \mathsf{perms}(\mathsf{H}_{\tilde{\mathsf{G}}})$ and $\exists \delta, \delta'$ such that $\Gamma \vdash \delta : \mathsf{p}[\mathsf{g}] \triangleright \emptyset$, $\Gamma \vdash \iota \otimes \delta' : \mathsf{t}[\mathsf{g}] \triangleright \Delta$ and $P \equiv \texttt{if}\ \delta' = \delta\ \texttt{then}\ P_1\ \texttt{else}\ P_2$.
(9) $\mathsf{identify}\{\mathsf{t}'\} \notin \mathsf{perms}(\mathsf{H}_{\tilde{\mathsf{G}}})$ and $\exists \delta, \delta'$ such that $\Gamma \vdash \iota \otimes \delta : \mathsf{t}'[\mathsf{g}] \triangleright \emptyset$, $\Gamma \vdash \_ \otimes \delta' : \mathsf{t}[\mathsf{g}] \triangleright \Delta$ and $P \equiv \texttt{if}\ \delta' = \delta\ \texttt{then}\ P_1\ \texttt{else}\ P_2$.
(10) $\mathsf{disseminate}\ \mathsf{G}\ \lambda \in \mathsf{perms}(\mathsf{H}_{\tilde{\mathsf{G}}})$, $\lambda \neq \omega$ and $\mathsf{countRef}(P, \Gamma, \mathsf{G}[\mathsf{t}[\mathsf{g}]]) > \lambda$.
(11) there exists a sub-hierarchy of H, $\mathsf{H}' = \mathsf{G}_k\{\tilde{\mathsf{prm}}\}[\mathsf{H}_i]_{i \in I}$ with $1 \leq k \leq n$ as well as $\mathsf{no\ dissemination}\langle\mathsf{kind}\rangle \in \tilde{\mathsf{prm}}$ and $\exists u$ such that $\Gamma \vdash u : \mathsf{G}'[\mathsf{G}[\mathsf{t}[\mathsf{g}]]] \triangleright \emptyset$, and $P \equiv u!\langle u'\rangle.\, P'$ with $\mathsf{G}' \notin \mathsf{groups}(\mathsf{H}')$.

According to the definition, the first five error systems require that the send or the receive prefixes inside a system do not respect $\mathcal{P}$. The first system is an error because it allows a input of private data inside a group hierarchy, where the corresponding group hierarchy in $\mathcal{P}$ does not include the read permission. Similarly, the second error system outputs private data and at the same time there is no update permission in the corresponding group hierarchy of $\mathcal{P}$. Systems of clauses (3) and (4) deal with input and output of reference name without the $\mathsf{reference}$ and $\mathsf{disseminate}\ \mathsf{G}\ \lambda$ permissions, respectively, in the corresponding group hierarchy in $\mathcal{P}$. The fifth error system requires that private data can be input along with access to its identity despite the lack of the readId permission in $\mathcal{P}$. The next error system defines a store process inside a group hierarchy with the store permission not in the corresponding policy

hierarchy. A system is an error with respect to aggregation if it aggregates stores of the same identity without the $\mathsf{aggregate}$ permission in the corresponding policy hierarchy (clause (7)). Private data usage is expressed via a matching construct: A usage error system occurs when a process inside a group hierarchy tries to match private data without the $\mathsf{usage\{p\}}$ permission in the policy (clause (8)). Similarly, an identification error process occurs when there is a matching on private data and there is no $\mathsf{identify\{t\}}$ permission defined by the policy (clause (9)). A system is an error with respect to a policy $\mathcal{P}$ if the number of output prefixes to references in its definition (counted with the $\mathsf{countRef}(P, \Gamma, t)$ function) are more than the λ in the $\mathsf{disseminate}\ G\ \lambda$ permission of the policy's hierarchy (clause (10)). Finally, if a policy specifies that no data should be disseminated outside some group G, then a process should not be able to send private data links to groups that are not contained within the hierarchy of G (clause (11)).

As expected, if a system is an error with respect to a policy $\mathcal{P}$ and an environment Γ then its Θ-interface does not satisfy policy $\mathcal{P}$:

Lemma 6.13. *Let system S be an error system with respect to a well-formed policy $\mathcal{P}$ and a type environment Γ. If $\Gamma \vdash S \triangleright \Theta$ then $\mathcal{P} \not\Vdash \Theta$.*

Sketch. The proof is based on the definition of error systems. Each error system S is typed as:

$$\Gamma; \Lambda \vdash S \triangleright \Theta$$

We then show that $\mathcal{P} \not\Vdash \Theta$.

The proofs for all error systems are similar. We give a typical case. Consider:

- System
$$S = \mathsf{G}_1[(\nu\ \tilde{x}_1)(\mathsf{G}_2[(\nu\ \tilde{x}_2)(\ldots(\mathsf{G}_n[(\nu\ \tilde{x}_n)(P \mid Q)] \| S_n)\ldots)])] \| S_1$$
with $P \equiv \overline{r} \triangleright [\mathsf{id} \otimes c]$;
- $\tilde{\mathsf{G}} = \mathsf{G}_1, \ldots \mathsf{G}_n$;
- Policy $\mathcal{P} = \mathsf{t} \gg \mathsf{H}; \mathcal{P}'$ such that $\mathsf{store} \notin \mathsf{perms}(\mathsf{H}_{\tilde{G}})$;
- Type environment Γ such that $\Gamma \vdash r : \mathsf{G[t[g]]} \triangleright \emptyset$.

We type process P using typing rule [TSt]:

$$[\mathsf{TSt}]\ \frac{\Gamma \vdash r : \mathsf{G[t[g]]} \triangleright \emptyset \quad \Gamma \vdash \mathsf{id} \otimes c : \mathsf{t[g]} \triangleright \emptyset}{\Gamma; r; \langle \mathsf{id}, \mathsf{t} \rangle \vdash \overline{r} \triangleright [\mathsf{id} \otimes c] \triangleright \mathsf{t} : \{\mathsf{store}\}}$$

We apply rule [TPar] to get:

$$[\mathsf{TPar}]\ \frac{\begin{array}{c}\Gamma; r; \langle \mathsf{id}, \mathsf{t} \rangle \vdash \overline{r} \triangleright [\mathsf{id} \otimes c] \triangleright \mathsf{t} : \{\mathsf{store}\} \\ \Gamma; \Lambda; Z \vdash Q \triangleright \Delta\end{array}}{\Gamma; \Lambda, r; Z, \langle \mathsf{id}, \mathsf{t} \rangle \vdash \overline{r} \triangleright [\mathsf{id} \otimes c] \mid Q \triangleright \Delta \uplus \mathsf{t} : \{\mathsf{store}\}}$$

We then apply type rules [TSRes], [TGr], [TSPar], [TSRes], and [TSGr], to get:

$$[\mathsf{TSGr}]\ \frac{\begin{array}{c}\ldots \\ \Gamma; \Lambda \vdash (\nu\ \tilde{x}_1)(\mathsf{G}_2[(\nu\ \tilde{x}_2)(\ldots(\mathsf{G}_n[(\nu\ \tilde{x}_n)(P \mid Q) \| S_n])\ldots)] \| S_1) \triangleright \Theta'\end{array}}{\Gamma; \Lambda \vdash S \vdash \Theta}$$

From the application of rules [TGr], [TSGr] we know that $\mathsf{t} : \mathsf{G}_1[\ldots \mathsf{G}_n[\tilde{\mathsf{prm}}]\ldots] \in \Theta$ for some $\tilde{\mathsf{prm}}$. From the definition of $\uplus$ and the application of rule [TPar] we know that $\mathsf{store} \in \tilde{\mathsf{prm}}$.

From this we can deduce: $\tilde{\mathsf{prm}} \not\preceq \mathsf{perms}(\mathsf{H}_{\tilde{G}}) \implies \mathsf{H} \not\Vdash \mathsf{G}_1[\ldots \mathsf{G}_n[\tilde{\mathsf{prm}}]\ldots] \implies \mathcal{P} \not\Vdash \Theta$, as required.

The remaining cases for the proof are similar. □

We may now conclude with a safety theorem which verifies that the satisfiability of a policy by a typed system is preserved by the semantics. Below we denote a (possibly empty) sequence of actions $\xrightarrow{l_1} \ldots \xrightarrow{l_n}$ with $\xRightarrow{\tilde{\ell}}$

Theorem 6.14 (Safety)**.** *If $\mathcal{P}; \Gamma \vdash S \triangleright \Theta$ and $S \xRightarrow{\tilde{\ell}} S'$ then S' is not an error with respect to policy $\mathcal{P}$.*

Proof. The proof is immediate by Corollary 6.10 and Lemma 6.13. □

7. Use Cases

7.1. **Electronic Traffic Pricing.** 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 determine the toll. To achieve this, for each vehicle detailed time and location information must be collected and processed and the due amount can be calculated with the help of a digital tariff and a road map. A number of possible implementation methods may be considered for this scheme [17]. In the centralised approach all location information is communicated to the pricing authority which computes the fee to be paid based on the received information. In the decentralised approach the fee is computed locally on the car via the use of a third trusted entity such as a smart card. In the following subsections we consider these approaches and their associated privacy characteristics.

7.2. **The centralised 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). 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 reliable.

We may model this system with the aid of five groups: ETP corresponds to the entirety of the ETP system, Car refers to the car and is divided into the OBE and the GPS subgroups, and PA refers to the pricing authority. Note that in our model we simplify the pricing scheme by omitting timing considerations. This, however, can be easily extended by the introduction of additional processes for storing timing information as well as the associated actions for storing/communicating the timings of the observations, similarly to the way locations are handled. Furthermore, according to this scheme, the car is responsible for frequently (e.g. once a day) sending its collected data log to the authority. We abstract away this behavior of the car, by modeling the car sending to the authority m random locations. Finally, we point out that in our model we simplify the study to a single car. Note that in order to reason about multiple cars we would need to define a distinct group Car_v for each vehicle v.

As far as types are concerned, we assume the existence of two ground types: λ referring to locations and ϕ referring to fees. Furthermore, we assume the private data types loc and fee and the constant type spotCheck. We write $T_l = \mathsf{loc}[\lambda]$, $T_f = \mathsf{fee}[\phi]$, $T_c^l = \mathsf{Car}[T_l]$, $T_{etp}^l = \mathsf{ETP}[T_l]$, $T_{pa}^l = \mathsf{PA}[T_l]$, $T_{pa}^f = \mathsf{PA}[T_f]$ and $T_{sc} = \mathsf{ETP}[\mathsf{ETP}[T_l]]$.

We define the system as follows:

$$
\begin{array}{rcl}
System & = & \mathsf{ETP}[\ (\nu\ spotcheck)(\nu\ topa)(\mathsf{Car}[C] \parallel \mathsf{PA}[A]))\ ] \\
C & = & (\nu\ r)(\overline{r} \triangleright [\mathsf{id} \otimes l] \ \parallel\ \mathsf{GPS}[L] \ \parallel\ \mathsf{OBE}[O]) \\
L & = & *\ lc?(newl).\, r!\langle \mathsf{id} \otimes newl \rangle.\, \mathbf{0} \\
O & = & *\ topa!\langle r \rangle.\, \mathbf{0} \ \mid\ *\ spotcheck?(z).\, z!\langle r \rangle.\, \mathbf{0} \\
A & = & (\nu\ r_1) \ldots (\nu\ r_m)(\overline{r_1} \triangleright [\mathsf{id} \otimes x_1] \mid \ldots \mid \overline{r_m} \triangleright [\mathsf{id} \otimes x_m] \mid \overline{f} \triangleright [\mathsf{id} \otimes y] \mid R \mid S) \\
R & = & topa?(z_1).\, z_1?(x \otimes y).\, r_1!\langle x \otimes y \rangle.\, \ldots \\
 & & topa?(z_m).\, z_m?(x \otimes y).\, r_m!\langle x \otimes y \rangle.\, f!\langle \mathsf{id} \otimes fee \rangle.\, \mathbf{0} \\
S & = & (\nu\ s) spotcheck!\langle s \rangle.\, s?(z).\, z?(x \otimes y).\, \mathtt{if}\ y = l_{sc}\ \mathtt{then}\ \mathbf{0}\ \mathtt{else}\ f!\langle \mathsf{id} \otimes fine \rangle.\, \mathbf{0}
\end{array}
$$

In the model above we have system $System$ where a car process C and an authority process A are nested within the ETP group. The two processes share names *topa* and *spotcheck* via which the car and the authority can communicate for delivering data from the car to the authority and for initiating spot checks, respectively. In turn, a car C is composed of a store process, where the identifier of the car is associated with the current location of the car, the component O belonging to group OBE, and the component L responsible for computing the current location belonging to group GPS. These three processes are nested within the Car group and behave as follows: The store is initiated with the identifier of the car and its initial location (l) and may be accessed by processes L and O. Process L computes the new locations of the car and saves them to the store. The OBE O may read the current location of the car from the store and forward it to the authority via channel *topa* or it may enquire this location as a response of a spot check initiated by the pricing authority A. In turn, authority A is defined as follows: it contains m stores for saving m consecutive locations of the car, a store for saving the fee of the car, and two processes R and S, which are responsible for computing the fee of the car and performing spot checks, respectively. Specifically, process R receives m consecutive locations of the car and saves them in its stores before calculating the fee based on the received values. This value, *fee*, is essentially obtained as a function of the values of the locations stored in the m stores of the process. In turn, process S performs a spot check on the car. During a spot check, component S creates a new channel s and sends it to the OBE via which the OBE is expected to return the current location for a verification check. This value is compared to the actual location of the spot check of the car, l_{sc}, and, if the location communicated by the car is erroneous, then the authority imposes a fine, *fine*.

By applying the rules of our type system we may show that $\Gamma; \emptyset \vdash System \triangleright \Theta$, where:

$$
\begin{array}{rcl}
\Gamma & = & \{lc : \mathsf{ETP}[\lambda], f : T^f_{pa}, l : \lambda, l_{sc} : \mathsf{spotCheck}[\lambda], fee : \phi, fine : \phi\} \\
\Theta & = & \mathsf{fee} : \mathsf{ETP}[\mathsf{PA}[\{\mathsf{store}, \mathsf{update}\}]], \\
 & & \mathsf{loc} : \mathsf{ETP}[\mathsf{Car}[\{\mathsf{store}\}]], \\
 & & \mathsf{loc} : \mathsf{ETP}[\mathsf{Car}[\mathsf{OBE}[\{\mathsf{disseminate\ ETP}\ \omega\}]]], \\
 & & \mathsf{loc} : \mathsf{ETP}[\mathsf{Car}[\mathsf{GPS}[\{\mathsf{update}\}]]], \\
 & & \mathsf{loc} : \mathsf{ETP}[\mathsf{PA}[\{\mathsf{reference}, \mathsf{store}, \mathsf{read}, \mathsf{readId}, \mathsf{usage}\{\mathsf{spotCheck}\}, \mathsf{aggregate}\}]]
\end{array}
$$

A possible privacy policy for the system might be one that states that locations may be freely forwarded by the OBE. We may define this by $\mathcal{P} = \mathsf{loc} \gg H$ where:

$$
\begin{array}{rcl}
H & = & \mathsf{ETP}\{\mathsf{no\ dissemination}\langle\mathsf{sensitive}\rangle\}[\\
& & \quad \mathsf{Car}\{\mathsf{store}\}[\mathsf{OBE}\{\mathsf{read}, \mathsf{readId}, \mathsf{disseminate\ ETP}\ \omega\}], \\
& & \quad \mathsf{GPS}\{\mathsf{read}, \mathsf{readId}, \mathsf{update}\}, \\
& & \quad \mathsf{PA}\{\mathsf{reference}, \mathsf{store}, \mathsf{read}, \mathsf{readId}, \mathsf{usage}\{\mathsf{spotCheck}\}, \mathsf{aggregate}\} \\
& &]
\end{array}
$$

By comparing environment Θ with policy $\mathcal{P}$, we may conclude that *System* satisfies $\mathcal{P}$.

This architecture is simple but also very weak in protecting the privacy of individuals: the fact that the PA gets detailed travel information about a vehicle constitutes a privacy and security threat. In our system this privacy threat can be pinpointed to private data type loc and the fact that references to locations may be communicated to the PA for an unlimited number of times. An alternative implementation that limits the transmission of locations is presented in the second implementation proposal presented below.

7.3. **The decentralised approach.** To avoid the disclosure of the complete travel logs of a car, the decentralised solution employs a third trusted entity (e.g. smart card) to make computations of the fee locally on the car and send its value 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. The new privacy policy might be $\mathcal{P} = \mathsf{loc} \gg H, \mathsf{fee} \gg F$ where:

$$
\begin{array}{rcl}
H & = & \mathsf{ETP}\ \{\mathsf{no\ dissemination}\langle\mathsf{sensitive}\rangle\}[\\
& & \quad \mathsf{Car}\ \{\mathsf{store}\}[\mathsf{OBE}\ \{\mathsf{disseminate\ ETP}\ 2\}], \\
& & \quad \mathsf{GPS}\ \{\mathsf{update}\}, \\
& & \quad \mathsf{SC}\ \{\mathsf{read}, \mathsf{readId}, \mathsf{usage}\{\mathsf{computeFee}\}\}, \\
& & \quad \mathsf{PA}\ \{\mathsf{reference}, \mathsf{read}, \mathsf{readId}, \mathsf{usage}\{\mathsf{spotCheck}\}\} \\
& &] \\
F & = & \mathsf{ETP}\ \{\mathsf{no\ dissemination}\langle\mathsf{sensitive}\rangle\}[\\
& & \quad \mathsf{Car}\ \{\}[\mathsf{OBE}\ \{\mathsf{disseminate\ ETP}\ 1\}], \\
& & \quad \mathsf{GPS}\ \{\}, \\
& & \quad \mathsf{SC}\ \{\mathsf{store}, \mathsf{update}, \mathsf{disseminate\ Car}\ \omega\}, \\
& & \quad \mathsf{PA}\ \{\mathsf{reference}, \mathsf{read}, \mathsf{readId}\} \\
& &]
\end{array}
$$

To model the new system as described above we assume a new group SC and a new component S which defines the code of a smart card, belonging to group SC:

$$\begin{array}{rcl}
System' & = & \mathsf{ETP}[\ (\nu\ spotcheck)(\nu\ topa)(\ \mathsf{Car}[C] \parallel \mathsf{PA}[A]\ )\ ] \\
C & = & (\nu\ r)\ (\ \overline{r} \triangleright [\mathsf{id} \otimes l]\ \parallel\ \mathsf{GPS}[L]\ \parallel\ \mathsf{OBE}[O]\ \parallel\ \mathsf{SC}[S]\ ) \\
L & = & *\ lc?(newl).\, r!\langle \mathsf{id} \otimes newl \rangle.\, \mathbf{0} \\
O & = & spotcheck?(z).\, z!\langle r \rangle.\, \mathbf{0} \\
 & | & send?(f).\, sendpa!\langle f \rangle.\, \mathbf{0} \\
S & = & (\nu\ f)(\overline{f} \triangleright [\mathsf{id} \otimes v] \mid (r?(x \otimes y).\,)^m f!\langle \mathsf{id} \otimes fee \rangle.\, send!\langle f \rangle.\, \mathbf{0}) \\
A & = & (\nu\ s)(spotcheck!\langle s \rangle.\, s?(r).\, r?(x \otimes y).\, \mathtt{if}\ y = l_{sc}\ \mathtt{then}\ \mathbf{0}\ \mathtt{else}\ f!\langle \mathsf{id} \otimes fine \rangle.\, \mathbf{0}) \\
 & | & sendpa?(x).\, x?(x \otimes y).\, \mathbf{0}
\end{array}$$

In this system, we point out that process S has a local store for saving the fee as this is computed after reading a number of location values from the car store. We write $(r?(x \otimes y).\,)^m P$ as a shorthand for prefixing the P process by m occurrences of the input label $r?(x \otimes y)$. As in the centralised approach, this is implemented as an abstraction of the actual behaviour of the system where, in fact, the complete log is communicated to the smart card for a specific time period. This could be captured more precisely in a timed extension of our framework. Moving on to the behaviour of S, once computing the fee, process S is responsible for announcing this fee to the OBE process which is then responsible for disseminating the fee to the Pricing Authority. When the PA receives a reference to the data it reads the fee assigned to the car. As a final point, we observe that the OBE process engages in exactly one spot check.

We may verify that $\Gamma'; \emptyset \vdash System' \triangleright \Theta'$, where $\Gamma' = \Gamma \cup \{send : \mathsf{Car}[T^f_{pa}], sendpa : \mathsf{ETP}[T^f_{pa}]\}$ and interface Θ' satisfies the enunciated policy.

7.4. **Privacy-preserving speed-limit enforcement.** Speed-limit enforcement is the action taken by appropriately empowered authorities to check that road vehicles are complying with the speed limit in force on roads and highways. A variety of methods are employed to this effect including automated roadside *speed-camera* systems based on Doppler-radar based measurements, radio-based transponder techniques, which require additional hardware on vehicles, or section control systems [41].

In most of these techniques, the process of detecting whether a vehicle has exceeded the speed limit is done centrally: data recorded is stored at a central server and processing of the data is implemented in order to detect violations. However, this practice clearly endangers the privacy of individuals and in fact goes against the law since the processing of personal data is prohibited unless there is evidence of a speed limit violation.

In our study below, we model an abstraction of the speed-camera scheme and an associated privacy policy and we show by typing that the model of the system satisfies the policy. Beginning with the requirements for privacy-preservation, the system should ensure the following:

(1) Any data collected by the speed cameras must only be used for detecting speed violations and any further processing is prohibited.
(2) Evidence data collected by speed cameras must not be stored permanently and must be destroyed immediately if no speed limit violation has been discovered. Storage beyond this point is only permitted in the case that a speed limit violation has been detected.

(3) Evidence data relating to the driver's identity must not be revealed unless a speed-limit violation is detected.

Specifically, in our model we consider a system (group SpeedControl) composed of car entities (group Car) and the speed-control authority (group SCSystem), itself composed of speed-camera entities (group trafficCam), the authority responsible for processing the data (group Auth) and the database of the system (group DBase) where the personal data of all vehicle owners are appropriately stored.

As far as types are concerned, we assume the existence of two ground types: Speed referring to vehicle speed and RegNum referring to vehicle registration numbers and three private data types CarReg, CarSpeed and DriverReg. Precisely, we consider CarReg to be the type of private data pertaining to a car's registration numbers as they exist on vehicles, CarSpeed to be the type of private data of the speed of vehicles and DriverReg to be the type of private data associating a driver's identity with the registration numbers of their car as they might exist on a system's database. Finally, we assume the constant type Limit: constants of type Limit may be compared against other data for the purpose of checking a speed limit violation.

The system should conform to the following policy:

$$
\begin{array}{rcl}
\mathsf{CarReg} & \gg & \mathsf{SpeedControl}\ \{\mathsf{no\ dissemination}\langle\mathsf{sensitive}\rangle\}[\\
& & \quad \mathsf{Car}\ \{\mathsf{store}, \mathsf{aggregate}, \mathsf{disseminate\ SpeedControl}\ \omega\}, \\
& & \quad \mathsf{SCSystem}\ \{\}[\\
& & \quad\quad \mathsf{trafficCam}\ \{\mathsf{reference}, \mathsf{disseminate\ SCSystem}\ \omega\}, \\
& & \quad\quad \mathsf{Auth}\ \{\mathsf{reference}, \mathsf{read}, \mathsf{aggregate}, \mathsf{identify}\{\mathsf{DriverReg}\}\}, \\
& & \quad\quad \mathsf{DBase}\ \{\} \\
& & \quad] \\
& &] \\
\mathsf{CarSpeed} & \gg & \mathsf{SpeedControl}\ \{\mathsf{no\ dissemination}\langle\mathsf{sensitive}\rangle\}[\\
& & \quad \mathsf{Car}\ \{\mathsf{update}, \mathsf{store}, \mathsf{aggregate}, \mathsf{disseminate\ SpeedControl}\ \omega\}, \\
& & \quad \mathsf{SCSystem}\ \{\}[\\
& & \quad\quad \mathsf{trafficCam}\ \{\mathsf{reference}, \mathsf{disseminate\ SCSystem}\ \omega\}, \\
& & \quad\quad \mathsf{Auth}\ \{\mathsf{reference}, \mathsf{read}, \mathsf{aggregate}, \mathsf{usage}\{\mathsf{Limit}\}, \mathsf{store}\}, \\
& & \quad\quad \mathsf{DBase}\ \{\} \\
& & \quad] \\
& &] \\
\mathsf{DriverReg} & \gg & \mathsf{SpeedControl}\ \{\mathsf{no\ dissemination}\langle\mathsf{sensitive}\rangle\}[\\
& & \quad \mathsf{Car}\ \{\} \\
& & \quad \mathsf{System}\ \{\}[\\
& & \quad\quad \mathsf{trafficCam}\ \{\}, \\
& & \quad\quad \mathsf{Auth}\ \{\mathsf{reference}, \mathsf{read}, \mathsf{readId}\}, \\
& & \quad\quad \mathsf{DBase}\ \{\mathsf{reference}, \mathsf{disseminate\ System}\ \omega, \mathsf{store}\} \\
& & \quad] \\
& &]
\end{array}
$$

According to the policy, data of type CarReg, corresponding to the registration number plate of a car, exist (are stored) in a car in public sight and thus can be disseminated within the speed-control system by the car for an unlimited number of times. A traffic camera, trafficCam, may thus gain access to a reference of this data (by taking a photo)

and disseminate such a CarReg reference inside the speed control SCSystem. The authority Auth may then receive this reference/photo and by various means read the actual CarReg data, which however remains anonymised unless the authority performs a check for the identification of the owner against other CarReg data. A database DBase does not have any permissions on data of type CarReg.

Data of type CarSpeed correspond to the speed of a Car and can be stored in a Car and updated during computation. Via a speed radar, CarSpeed can be disseminated. A trafficCam may thus read the speed of a car and disseminate it (along with any other accompanying information such as a photograph of the car) inside the speed control SCSystem. An authority Auth may read the anonymised Car speed and use it for the purpose of checking speed violation. A database DBase does not have any permissions on data of type CarSpeed.

Finally, a DriverReg corresponds to the association of a registration number and a driver inside a database. Car and trafficCam have no permissions on such data. An authority Auth may read the data DriverReg with their identity. A database system DBase stores these data and disseminates them inside the speed control system.

We model an abstraction of the speed-camera scheme as follows. Note that for the purposes of the example we have enriched our calculus with operator $<$. This extension can easily be incorporated into our framework.

$$
\begin{array}{rcl}
\mathit{System} & = & \mathsf{SpeedControl}[\ \mathsf{Car}[C] \parallel \mathsf{SCSystem}[\mathsf{trafficCam}[SC] \parallel \mathsf{Auth}[A] \parallel \mathsf{DBase}[D]\ ]] \\
C & = & (\nu\ r)(\nu\ s)(\overline{r} \triangleright [\mathsf{id} \otimes \mathit{reg}] \mid \overline{s} \triangleright [\mathsf{id} \otimes \mathit{speed}] \\
 & \mid & *\mathit{cs}?(y).\, s!\langle \mathsf{id} \otimes y \rangle.\, \mathbf{0} \\
 & \mid & *p!\langle r, s \rangle.\, \mathbf{0}) \\
SC & = & *p?(x, y).\, a!\langle x, y \rangle.\, \mathbf{0} \\
A & = & *a?(k_1, k_2).\, k_2?(\_ \otimes z).\, \mathtt{if}\ (z > \mathit{overLim})\ \mathtt{then}\ V\ \mathtt{else}\ \mathbf{0} \\
V & = & k_1?(\_ \otimes y). \\
 & & (r_1?(x \otimes w).\, \mathtt{if}\ w = y\ \mathtt{then}\ (\nu\ r)(\overline{r} \triangleright [x \otimes z])\ \mathtt{else}\ \mathbf{0} \\
 & \mid & \ldots \\
 & \mid & r_n?(x \otimes w).\, \mathtt{if}\ w = y\ \mathtt{then}\ (\nu\ r)(\overline{r} \triangleright [x \otimes z])\ \mathtt{else}\ \mathbf{0}) \\
D & = & \overline{r_1} \triangleright [\mathsf{id}_1 \otimes \mathsf{reg}_1] \mid \ldots \mid \overline{r_n} \triangleright [\mathsf{id}_n \otimes \mathsf{reg}_n]
\end{array}
$$

In system *System* we have a car process C, an authority process A, a speed camera SC, and a database D nested within the SpeedControl group and sharing:

- name p between the car and the speed camera via which a photo of the car (registration number) and its speed are collected by the speed camera,
- name a via which this information is communicated from the speed camera to the speed-enforcement authority, and
- references $r_1, \ldots, r_n$, via which the authority queries the database to extract the identity of the driver corresponding to the registration number of the car that has exceeded the speed limit.

According to the definition, a car C possesses two stores containing its registration number and its speed, with the speed of the car changing dynamically, as modelled by input on channel cs. These two pieces of information can be obtained by the speed camera via channel p. On receipt of this information, the speed camera forwards the information to the authority A. In turn, authority A may receive such information from the speed camera and process

it as follows: It begins by accessing the speed of the car. Note that the identity of the car driver is hidden. It then proceeds to check whether this speed is above the speed limit. This is achieved by comparing the vehicle speed with value *overLim* which captures unacceptable speed values. In case a violation is detected, the authority proceeds according to process V, and the authority communicates with the database in order to find out the identity of the driver associated with the speed limit violation. The identification is performed by matching the anonymised registration number of the vehicle against the records that are received by the database. Finally, the database, stores all information about the registration number of all the drivers.

Let us write $T_{cr} = \mathsf{CarReg}[\mathsf{RegNum}]$, $T_{sp} = \mathsf{CarSpeed}[\mathsf{Speed}]$, $T_{dr} = \mathsf{DriverReg}[\mathsf{RegNum}]$, $T^S_{cr} = \mathsf{SpeedControl}[T_{cr}]$, $T^S_{sp} = \mathsf{SpeedControl}[T_{sp}]$, $T^S_{dr} = \mathsf{SCSystem}[T_{dr}]$. By applying the rules of our type system we may show that $\Gamma; \emptyset \vdash \mathsf{System} \triangleright \Theta$, where:

$$
\begin{array}{rcl}
\Gamma & = & \{\mathit{overLim} : \mathsf{Limit}[\mathsf{Speed}], \\
& & p, a : \mathsf{SpeedControl}[T_{cr}, T_{sp}], r_1, \ldots, r_n : T^S_{dr}\} \\
\Theta & = & \mathsf{CarReg} : \mathsf{SpeedControl}[\mathsf{Car}[\{\mathsf{store}, \mathsf{aggregate}, \mathsf{disseminate\ SpeedControl}\ \omega\}]], \\
& & \mathsf{CarReg} : \mathsf{SpeedControl}[\mathsf{SCSystem}[\mathsf{trafficCam}[\{\mathsf{reference}, \mathsf{disseminate\ SpeedControl}\ \omega\}]]], \\
& & \mathsf{CarReg} : \mathsf{SpeedControl}[\mathsf{SCSystem}[\mathsf{Auth}[\{\mathsf{reference}, \mathsf{read}, \mathsf{aggregate}, \mathsf{identify}\{\mathsf{DriverReg}\}\}]]], \\
& & \mathsf{CarReg} : \mathsf{SpeedControl}[\mathsf{SCSystem}[\mathsf{DBase}[\{\}]]] \\
& & \mathsf{Speed} : \mathsf{SpeedControl}[\mathsf{Car}[\{\mathsf{update}, \mathsf{store}, \mathsf{aggregate}, \mathsf{disseminate\ SpeedControl}\ \omega\}]], \\
& & \mathsf{Speed} : \mathsf{SpeedControl}[\mathsf{SCSystem}[\mathsf{trafficCam}[\{\mathsf{reference}, \mathsf{disseminate\ SpeedControl}\ \omega\}]]], \\
& & \mathsf{Speed} : \mathsf{SpeedControl}[\mathsf{SCSystem}[\mathsf{Auth}[\{\mathsf{reference}, \mathsf{read}, \mathsf{aggregate}, \mathsf{usage}\{\mathsf{Limit}\}\}]]], \\
& & \mathsf{Speed} : \mathsf{SpeedControl}[\mathsf{SCSystem}[\mathsf{DBase}[\{\}]]] \\
& & \mathsf{DriverReg} : \mathsf{SpeedControl}[\mathsf{Car}[]], \\
& & \mathsf{DriverReg} : \mathsf{SpeedControl}[\mathsf{SCSystem}[\mathsf{trafficCam}[]]], \\
& & \mathsf{DriverReg} : \mathsf{SpeedControl}[\mathsf{SCSystem}[\mathsf{Auth}[\{\mathsf{reference}, \mathsf{read}, \mathsf{readId}\}]]], \\
& & \mathsf{DriverReg} : \mathsf{SpeedControl}[\mathsf{SCSystem}[\mathsf{DBase}[\{\mathsf{store}, \mathsf{disseminate\ SCSystem}\ \omega\}]]]
\end{array}
$$

We may prove that Θ is compatible with the enunciated policy; therefore, the policy is satisfied by the system.

8. 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 [14, 3, 36, 23, 35, 38, 11]. 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 [35, 30], on-the-fly using monitoring, e.g. [5, 42], or *a posteriori*, e.g. through audit procedures [16, 4, 19].

Among these studies we mention work on the notion of Contextual Integrity [4], which constitutes a philosophical account of privacy in terms of the transfer of personal information. Aspects of this framework have been formalised in a logical framework and were used for specifying privacy regulations like HIPAA while notions of compliance of policies by systems were considered. Close to our work is also the work of Ni *et al.* [39] where a family of models named P-RBAC (Privacy-aware Role Based Access Control) is presented that extends the

traditional role-based access control to support specification of complex privacy policies. This model is based on the concepts of roles, permissions and data types, similar to ours, but it may additionally specify conditions, purposes and obligations. The methodology is mostly geared towards expressing policies and checking for conflicts within policies as opposed to assessing the satisfaction of policies by systems, which is the goal of our work. Finally, we point out that the notion of policy hierarchies is closely related to the concept of hierarchical P-RBAC of [39]. Hierarchical P-RBAC introduces, amongst others, the notion of role hierarchies often present in extensions to RBAC. Role hierarchies are a natural means for structuring roles to reflect an organisation's lines of authority and responsibility. Nonetheless, all these works focus on privacy-aware *access control*. Our work extends these approaches by considering *privacy* as a general notion and addressing a wider set of privacy violations such as identification and aggregation.

Also related to our work is the research line on type-based security in process calculi. Among these works, numerous studies 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 [25, 26, 24]. Furthermore, discretionary access control has been considered in [8] which similarly to our work employs the π-calculus with groups, while role-based access control (RBAC) has been considered in [7, 20, 13]. In addition, authorization policies and their analysis via type checking has been considered in a number of papers including [22, 2, 6]. Our work is similar in spirit to these works: all approaches consider some type of policy/schema and a type system that ensures that systems comply to their policy. Futhermore, we mention that a type system for checking differential privacy for security protocols was developed in [21] for enforcing quantitative privacy properties.

These works, however, differ from the work presented in this paper. To begin with, our approach departs from these works in that our study introduces the concepts of an attribute, hierarchies of disclosure zones and by basing the methodology around the problem of policy satisfaction. Furthermore, our goal is to provide foundations for a more general treatment of privacy which departs from access-control requirements. Inspired by Solove's taxonomy, we propose a framework for reasoning about identification, data aggregation and secondary use. To the best of our knowledge, our work is the first formal study of these notions.

The Privacy calculus together with the proposed type system can be used as a basis for future tool and type checker implementation. In this arena we can find a number of type checkers inspired by process calculi. In [15] a featherweight Scala type-checking model is proposed. The model is able to capture basic Scala constructs such as nested classes, abstract types, mixin composition, and path dependent types. In correspondence with the Privacy calculus we can benefit from nested classes to represent a Policy/System hierarchy. A second work that implements a distributed cryptographic asynchronous π-calculus can be found in [1] which is designed to enforce notions of secrecy. The implementation uses an intermediate language that implements a typed distributed cryptographic π-calculus. In our setting the notion of secrecy is related with the existence of groups. Recently, there has been an effort by the behavioural types community to implement π-calculus inspired type-checking algorithms [28, 31, 18, 40] as a variety of tools operating in different platforms. This line of work provides evidence that π-calculus type systems are volatile enough to provide a basis for mainstream programming.

To conclude, we mention our previous work of [32, 33, 29]. In these works we again employed the π-calculus with groups [10] accompanied by a type system for capturing

privacy-related notions. The type system of [32] was based on i/o and linear types for reasoning about information processing and dissemination and a two-level type structure for distinguishing between the permissions associated with a name upon receipt and upon delivery by a process. In [33], the type system was reconstructed and simplified: the notion of groups was employed to distinguish between the different entities of a system and we employ the type system in order to type-check a system against the standard types of the π-calculus with groups while performing type inference to associate permissions with the different components of a system. Furthermore, a safety criterion was proved for the framework thus providing the necessary machinery for proving privacy preservation by typing. Finally, in [29] we extended the results of [33] to encompass the notion of a purpose. In the present paper, we extend these works providing a more thorough treatment of privacy violations, including data identification, aggregation and secondary use. To achieve this, it was necessary to extend our policy language to include a wider range of privacy requirements and also to enrich both the underlying calculus as well as the associated type system in order to capture privacy-related concepts in a more satisfactory manner. Furthermore, the current paper contains the complete exposition of the methodology, including the proofs of all results.

9. 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 Γ and produces a permission interface Θ which satisfies a policy $\mathcal{P}$, then Sys complies to $\mathcal{P}$. Consequently, whenever a system type checks against a typing that is compatible with a privacy policy we may conclude that the system satisfies the policy.

The approach we have proposed is to a large extent an orthogonal extension of the selected framework, the π-calculus with groups. Modelling a system and constructing its related types is developed by using standard process-calculus techniques without the need of considering privacy matters, other than the scoping of groups. Then, the actual treatment of privacy within our framework takes place at the level of privacy policies against which a system should comply. The policy language we have proposed is a simple language that constructs a hierarchical structure of the entities composing a system and assigning permissions for accessing sensitive data to each of the entities while allowing to reason about possible privacy violations. These permissions are certainly not intended to capture every possible privacy issue, but rather to demonstrate a method of how one might formalise privacy rights. Identifying an appropriate and complete set of permissions for providing foundations for the notion of privacy in the general context should be the result of intensive and probably interdisciplinary research that justifies each choice.

To this effect, Solove's taxonomy of privacy violations forms a promising context in which these efforts can be based and it provides various directions for future work. One possible extension would be to add semantics both at the level of our metatheory as well as our policy language to capture *information-processing*-related privacy violations such as *distortion* and *insecurity* violations. Distortion allows for relating false information to a data subject and insecurity violations take place when some adversary steals an identity and poses as the data subject.

As a long-term goal, a possible application of such work would be to implement type checkers for statically ensuring that programs do not violate privacy policies. For such an effort to have merit various aspects need to been taken into account. To begin with, the machinery employed needs to be carefully designed, in co-operation with legal scholars and consultants, in order to guarantee the appropriateness and completeness of the methodology with respect to actual violations in the real world. Furthermore, one should address the question whether privacy-policy type checking would result in programs that would not create any legal implications both for the owners and the users of the program. Last but not least, the fact that privacy itself is subjective introduces the notion of consensus among communities from different disciplines on definition(s) of privacy policies and policy compliance.

Other possible directions for extending our work can be inspired by existing work on privacy within e.g. contextual integrity and privacy-aware RBAC as well as k-anonymity. For instance, we are currently extending our work in order to reason about more complex privacy policies that include *conditional* permissions and the concept of an *obligation*. Finally, it would be interesting to explore more dynamic settings where the roles held by an agent may evolve over time.

References

[1] Martín Abadi, Ricardo Corin, and Cédric Fournet. Computational secrecy by typing for the pi calculus. In *Proceedings of APLAS'06*, LNCS 4279, pages 253–269. Springer, 2006.

[2] Michael Backes, Catalin Hritcu, and Matteo Maffei. Type-checking zero-knowledge. In *Proceedings of CCS'08*, pages 357–370, 2008.

[3] Michael Backes, Birgit Pfitzmann, and Matthias Schunter. A toolkit for managing enterprise privacy policies. In *Proceedings of ESORICS'03*, LNCS 2808, pages 162–180. Springer, 2003.

[4] Adam Barth, Anupam Datta, John C. Mitchell, and Helen Nissenbaum. Privacy and contextual integrity: Framework and applications. In *Proceedings of S&P'06*, pages 184–198, 2006.

[5] David A. Basin, Felix Klaedtke, and Samuel Müller. Policy monitoring in first-order temporal logic. In *Proceedings of CAV'10*, LNCS 6174, pages 1–18. Springer, 2010.

[6] Jesper Bengtson, Karthikeyan Bhargavan, Cédric Fournet, Andrew D. Gordon, and Sergio Maffeis. Refinement types for secure implementations. *ACM Transactions on Programming Languages and Systems*, 33(2):8:1– 8:45, 2011.

[7] Chiara Braghin, Daniele Gorla, and Vladimiro Sassone. Role-based access control for a distributed calculus. *Journal of Computer Security*, 14(2):113–155, 2006.

[8] Michele Bugliesi, Dario Colazzo, Silvia Crafa, and Damiano Macedonio. A type system for discretionary access control. *Mathematical Structures in Computer Science*, 19(4):839–875, 2009.

[9] Ji-Won Byun, Elisa Bertino, and Ninghui Li. Purpose based access control of complex data for privacy protection. In *Proceedings of SACMAT'05*, pages 102–110. ACM, 2005.

[10] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Secrecy and group creation. *Information and Computation*, 196(2):127–155, 2005.

[11] Omar Chowdhury, Andreas Gampe, Jianwei Niu, Jeffery von Ronne, Jared Bennatt, Anupam Datta, Limin Jia, and William H. Winsborough. Privacy promises that can be kept: a policy analysis method with application to the HIPAA privacy rule. In *Proceedings of SACMAT'13*, pages 3–14, 2013.

[12] Pietro Colombo and Elena Ferrari. Enforcement of purpose based access control within relational database management systems. *IEEE Transactions on Knowledge and Data Engineering*, 26(11):2703–2716, 2014.

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

[14] Lorrie Faith Cranor. *Web privacy with P3P - the platform for privacy preferences*. O'Reilly, 2002.

[15] Vincent Cremet, François Garillot, Sergueï Lenglet, and Martin Odersky. A core calculus for Scala type checking. In *Proceedings of MFCS'06*, LNCS 4162, pages 1–23. Springer, 2006.

[16] Anupam Datta, Jeremiah Blocki, Nicolas Christin, Henry DeYoung, Deepak Garg, Limin Jia, Dilsun Kirli Kaynar, and Arunesh Sinha. Understanding and protecting privacy: Formal semantics and principled audit mechanisms. In *Proceedings of ICISS'11*, pages 1–27, 2011.
[17] Wiebren de Jonge and Bart Jacobs. Privacy-friendly electronic traffic pricing via commits. In *Proceedings of FAST'08*, LNCS 5491, pages 143–161. Springer, 2009.
[18] Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. Practical interruptible conversations: distributed dynamic verification with multiparty session types and python. *Formal Methods in System Design*, 46(3):197–225, 2015.
[19] Henry DeYoung, Deepak Garg, Limin Jia, Dilsun Kirli Kaynar, and Anupam Datta. Experiences in the logical specification of the HIPAA and GLBA privacy laws. In *Proceedings of WPES'10*, pages 73–82, 2010.
[20] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Svetlana Jaksic, and Jovanka Pantovic. Types for role-based access control of dynamic web data. In *Proceedings of WFLP'10*, LNCS 6559, pages 1–29. Springer, 2010.
[21] Fabienne Eigner and Matteo Maffei. Differential privacy by typing in security protocols. In *Proceedings of CSF'13*, pages 272–286, 2013.
[22] Cédric Fournet, Andy Gordon, and Sergio Maffeis. A type discipline for authorization in distributed systems. In *Proceedings of CSF'07*, pages 31–48, 2007.
[23] Deepak Garg, Limin Jia, and Anupam Datta. Policy auditing over incomplete logs: theory, implementation and applications. In *Proceedings of CCS'08*, pages 151–162, 2011.
[24] Matthew Hennessy. *A distributed Pi-calculus*. Cambridge University Press, 2007.
[25] Matthew Hennessy, Julian Rathke, and Nobuko Yoshida. safedpi: a language for controlling mobile code. *Acta Informatica*, 42(4-5):227–290, 2005.
[26] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120, 2002.
[27] Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *Proceedings of ESOP'98*, LNCS 1381, pages 122–138. Springer, 1998.
[28] Raymond Hu and Nobuko Yoshida. Hybrid session verification through endpoint API generation. In *Proceedings of FASE'16*, LNCS 9633, pages 401–418. Springer, 2016.
[29] Eleni Kokkinofta and Anna Philippou. Type checking purpose-based privacy policies in the π-calculus. In *Proceedings of BEAT/WS-FM'15*, LNCS 9421, pages 122–142. Springer, 2015.
[30] Masoud Koleini, Eike Ritter, and Mark Ryan. Model checking agent knowledge in dynamic access control policies. In *Proceedings of TACAS'13*, LNCS 7795, pages 448–462. Springer, 2013.
[31] Dimitrios Kouzapas, Ornela Dardha, Roly Perera, and Simon J. Gay. Typechecking protocols with Mungo and StMungo. In *Proceedings of PPDP'16*, pages 146–159, 2016.
[32] Dimitrios Kouzapas and Anna Philippou. A typing system for privacy. In *Proceedings of SEFM Workshops 2013*, LNCS 8368, pages 56–68. Springer, 2014.
[33] Dimitrios Kouzapas and Anna Philippou. Type checking privacy policies in the *pi*-calculus. In *Proceedings of FORTE'15*, LNCS 9039, pages 181–195. Springer, 2015.
[34] Marc Langheinrich. Privacy by design - principles of privacy-aware ubiquitous systems. In *Proceedings of UBICOMP'01*, LNCS 2201, pages 273–291. Springer, 2001.
[35] Ying Liu, Samuel Müller, and Ke Xu. A static compliance-checking framework for business process models. *IBM Systems Journal*, 46(2):335–362, 2007.
[36] Michael J. May, Carl A. Gunter, and Insup Lee. Privacy APIs: Access control techniques to analyze and verify legal privacy policies. In *Proceedings of CSFW'06*, pages 85–97, 2006.
[37] George C. Necula. Proof-carrying code. In *Proceedings of POPL'97*, pages 106–119. ACM Press, 1997.
[38] Qun Ni, Elisa Bertino, and Jorge Lobo. An obligation model bridging access control policies and privacy policies. In *Proceedings of SACMAT'08*, pages 133–142, 2008.
[39] Qun Ni, Elisa Bertino, Jorge Lobo, Carolyn Brodie, Clare-Marie Karat, John Karat, and Alberto Trombetta. Privacy-aware role-based access control. *ACM Trans. on Information and System Security*, 13(3):24–31, 2010.
[40] Frank Pfenning, Lus Caires, Bernardo Toninho, and Dennis Griffith. From linear logic to session-typed concurrent programming. Tutorial at POPL 2015, January 2015.

[41] Stefan Rass, Peter Schartner, Patrick Horster, and Alexander Abl. Privacy-preserving speed-limit enforcement. *Journal of Traffic and Logistics Engineering*, 2(1):26–33, 2014.
[42] Oleg Sokolsky, Usa Sammapun, Insup Lee, and Jesung Kim. Run-time checking of dynamic properties. *Electronic Notes Theoretical Computer Science*, 144(4):91–108, 2006.
[43] Daniel J. Solove. A Taxonomy of Privacy. *University of Pennsylvania Law Review*, 154(3):477–560, 2006.
[44] Michael Carl Tschantz, Anupam Datta, and Jeannette M. Wing. Formalizing and enforcing purpose restrictions in privacy policies. In *Proceedings of SP'12*, pages 176–190. IEEE Computer Society, 2012.
[45] Michael Carl Tschantz and Jeannette M. Wing. Formal methods for privacy. In *Proceedings of FM'09*, LNCS 5850, pages 1–15. Springer, 2009.

Appendix A. Encoding

For our calculus we propose the syntax and the semantics of a class of higher-level processes, notably the store process $\overline{r} \triangleright [\mathsf{id} \otimes c]$. Nevertheless, the syntax and the interaction of the store process can be fully encoded in terms of the standard π-calculus extended with the *selection and branch* syntax and semantics.

Definition A.1 (Encoding to the π-calculus)**.** Figure 8 defines the encoding from the Privacy calculus to the π-calculus. The rest of the operators are defined homomorphically.

$$
\begin{array}{rcl}
(\!|\overline{r} \triangleright [\mathsf{id} \otimes c]|\!) & \stackrel{\mathtt{def}}{=} & (\nu\ a)(a!\langle \mathsf{id} \otimes c\rangle.\,\mathbf{0} \mid a?(x \otimes y).\, r?(l). \\
& & \qquad l \triangleright \left\{ \begin{array}{lcl} \mathsf{rd} & : & l!\langle x \otimes y\rangle.\, a!\langle x \otimes y\rangle.\,\mathbf{0}, \\ \mathsf{wr} & : & l?(w \otimes z). \\ & & \mathtt{if}\ w = \mathsf{id}\ \mathtt{then}\ l \triangleleft \mathsf{ok}.\, a!\langle w \otimes z\rangle.\,\mathbf{0} \\ & & \mathtt{else}\ l \triangleleft \mathsf{fail}.\, a!\langle x \otimes y\rangle.\,\mathbf{0} \end{array} \right\}) \\
(\!|u?(k).\, P|\!) & \stackrel{\mathtt{def}}{=} & (\nu\ a)(u!\langle a\rangle.\, a \triangleleft \mathsf{rd}.\, a?(k).\, (\!|P|\!)) \qquad k \neq x \\
(\!|u!\langle \iota \otimes \delta\rangle.\,|\!)P & \stackrel{\mathtt{def}}{=} & (\nu\ a)((\nu\ b)(b!\langle \iota \otimes \delta\rangle.\,\mathbf{0} \mid\ *\, b?(k).\, (\nu\ e)(u!\langle e\rangle.\, e \triangleleft \mathsf{wr}.\, e!\langle k\rangle. \\
& & e \triangleright \{\mathsf{ok} : \overline{a}.\,\mathbf{0}, \mathsf{fail} : b!\langle k\rangle.\,\mathbf{0}\})) \mid a.\, (\!|P|\!))
\end{array}
$$

Figure 8: Encoding of the store process from the Privacy calculus into the standard π-calculus

Figure 8 presents the encoding of the store process syntax and semantics. A store process is represented as a recursive process that receives a name y and subsequently offers the choices of read (rd) and write (wr) on y. A client of the store process will make a selection between the choices for read and write. In the case of rd, the store simply sends the private data to the client and continues by recursion to its original state. In the case of wr, the store will receive data from the client and store them, given that the data have the correct id. If the data do not have the correct identity the interaction does not take place and the store continues by recursion to its previous state.

An input on a reference channel r is encoded with the creation of a new name a that is subsequently being sent via r to the store process. It then sends the rd label on channel a and receives from the store the private data value.

An output on a reference channel r is also encoded with the creation of a new name a that is subsequently sent via channel r to the store process. In contrast to the input encoding it will send the wr label on channel a and then send the private data to the store.

The store will then either reply with a success via label ok whereby the process will continue with the continuation $(\!|P|\!)$, or it will reply with fail whereby the process will use recursion to continue to its starting state.

The next theorem shows that the encoding enjoys sound and complete operational correspondence.

Theorem A.2 (Operational Correspondence)**.** *Let P be a process constructed on the process terms of the π-calculus without the store and extended with the selection/branch construct.*

(i) *If $P \longrightarrow P'$ then $(\!|P|\!) \longrightarrow (\!|P'|\!)$.*
(ii) *If $(\!|P|\!) \longrightarrow Q$ then either $Q \Longrightarrow P$, or there exists P' such that $P \longrightarrow P'$ and $Q \longrightarrow (\!|P'|\!)$.*

Proof. The proof for part (i) is done by induction on the structure of transition $\longrightarrow$. There are two interesting base cases:

- Case: $r?(x \otimes y).\, P \,|\, \overline{r} \triangleright [\mathsf{id} \otimes c] \longrightarrow P\{^{\mathsf{id} \otimes c}/_{x \otimes y}\} \,|\, \overline{r} \triangleright [\mathsf{id} \otimes c]$
- Case: $r!\langle \mathsf{id} \otimes c' \rangle.\, P \,|\, \overline{r} \triangleright [\mathsf{id} \otimes c] \longrightarrow P \,|\, \overline{r} \triangleright [\mathsf{id} \otimes c']$

The requirements of part (i) can be easily verified following simple transitions.

The proof for part (ii) is done by induction on the cases where $(\!|P|\!) \longrightarrow$ Q. The interesting cases are the base cases:

- $(\!|r?(x \otimes y).\, P \,|\, \overline{r} \triangleright [\mathsf{id} \otimes c]|\!) \longrightarrow Q$
  We can verify that $Q \Longrightarrow (\!|P\{^{\mathsf{id} \otimes c}/_{x \otimes y}\} \,|\, \overline{r} \triangleright [\mathsf{id} \otimes c]|\!)$ with simple transitions.
- $(\!|r!\langle \mathsf{id} \otimes c' \rangle.\, P \,|\, \overline{r} \triangleright [\mathsf{id} \otimes c]|\!) \longrightarrow Q$
  We can verify that $Q \Longrightarrow (\!|\overline{r} \triangleright [\mathsf{id} \otimes c] \longrightarrow P \,|\, \overline{r} \triangleright [\mathsf{id} \otimes c']|\!)$ with simple transitions.
- $(\!|r!\langle \mathsf{id}' \otimes c' \rangle.\, P \,|\, \overline{r} \triangleright [\mathsf{id} \otimes c]|\!) \longrightarrow Q$ with $\mathsf{id}' \neq \mathsf{id}$.
  We can verify that $Q \Longrightarrow (\!|r!\langle \mathsf{id}' \otimes c' \rangle.\, P \,|\, \overline{r} \triangleright [\mathsf{id} \otimes c]|\!)$ with simple transitions. □