Privacy by Design: a Formal Framework for the Analysis of Architectural Choices (extended version)

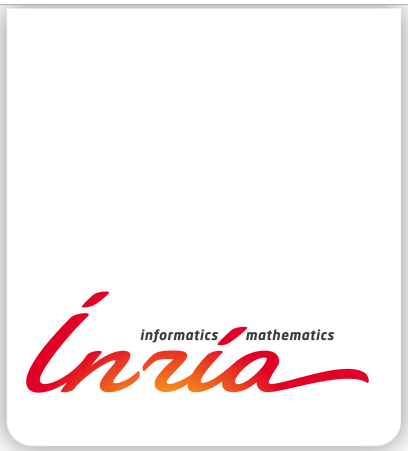
Daniel Le Métayer

Privacy by Design: a Formal Framework for the Analysis of Architectural Choices (Extended Version)

Daniel Le Métayer

RESEARCH REPORT
N° 8229
February 2013
Project-Team Privatics

ISSN 0249-6399 ISRN INRIA/RR--8229--FR+ENG



Privacy by Design: a Formal Framework for the Analysis of Architectural Choices (Extended Version)

Daniel Le Métayer

Project-Team Privatics

Research Report n° 8229 — February 2013 — 23 pages

Abstract: The privacy by design approach has already been put into practice in different application areas. We believe that the next challenge today is to go beyond individual cases and to provide methodologies to explore the design space in a systematic way. As a first step in this direction, we focus in this report on the data minimization principle and consider different options using decentralized architectures in which actors do not necessarily trust each other. We propose a framework to express the parameters to be taken into account (the service to be performed, the actors involved, their respective requirements, etc.) and an inference system to derive properties such as the possibility for an actor to detect potential errors (or frauds) in the computation of a variable. This inference system can be used in the design phase to check if an architecture meets the requirements of the parties or to point out conflicting requirements.

Key-words: privacy, design, architecture, methodology, formal, model

Author's address: Inria Grenoble Rhône-Alpes, Université de Lyon, CITI, Domaine Scientifique de la Doua, Bâtiment Claude Chappe, 6 avenue des Arts, 69621 Villeurbanne, France

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Protection de la vie privée: un cadre formel pour analyser les choix d'architecture (version étendue)

Résumé : La démarche de protection de la vie privée par conception (ou "privacy by design") a déjà été mise en pratique dans différents domaines d'applications. Le prochain défi en la matière est de dépasser le traitement au cas par cas pour fournir des méthodes de conception plus systématiques. Dans ce rapport, nous proposons à cet effet une méthode mettant en oeuvre le principe de minimisation des données. Elle permet d'analyser différents choix de conception reposant sur des architectures décentralisées dans lesquelles les acteurs ne s'accordent pas forcément une totale confiance. Le cadre proposé permet d'exprimer les paramètres à prendre en compte (service à assurer, acteurs impliqués, exigences en terme de protection des données ou d'accès aux informations, etc.) et d'analyser les choix d'architectures à l'aide d'un système d'inférence. Ce système peut être utilisé dans la phase de conception pour montrer qu'une architecture satisfait toutes les propriétés requises ou pour détecter des exigences inconciliables.

Mots-clés : vie privée, donnée personnelle, conception, architecture, methodologie, formel, modèle

1 Motivation

The privacy by design approach is often praised by lawyers as well as computer scientists as an essential step towards a better privacy protection [28, 45]. The general philosophy of privacy by design is that privacy should not be treated as an afterthought but rather as a first-class requirement during the design of a system. The approach has been applied in different areas such as smart metering [18, 32, 47], electronic traffic pricing [2, 22, 44]), ubiquitous computing [28] or location based services [14, 25, 26]. More generally, it is possible to identify a number of core principles that are widely accepted and can form a basis for privacy by design. For example, the Organization for Economic Co-operation and Development (OECD) has put forward principles [43] such as the consent, limitation of use, data quality, security and accountability. These principles, which were themselves inspired by the fair information practices initially proposed by a U.S. government advisory committee in the seventies, are also in line with the European Directive 95/46/EC on data protection.

One must admit however that the take-up of privacy by design in the industry is still rather limited. This situation is partly due to legal and economic reasons: as long as the law does not impose binding commitments, ICT providers and data collectors do not have sufficient incentives to invest into privacy by design [30]. The situation on the legal side might change in Europe though because the regulation proposed by the European Commission in January 2012 (to replace the European Directive 95/46/EC) includes binding commitments on privacy by design[1].

But the reasons for the lack of adoption of privacy by design are not only legal and economic [7]: even though computer scientists have devised a wide range of privacy enhancing tools [15, 19, 46], no general methodology is available to integrate them in a consistent way to meet a set of privacy requirements. Indeed, privacy by design goes beyond the use of privacy enhancing tools : it has to do with the general requirements of a system and the definition of its architecture. As such, privacy by design is a matter of choice: multiple options are generally available to achieve a given set of functionalities, some of them being privacy friendly, others less. Therefore, it is necessary to have a clear view of the overall system, the actors involved, what they need to know and the information flows between them, in order to ensure that a given choice of tools is consistent with the privacy requirements.

The next challenge in this area is thus to go beyond individual cases and to establish sound foundations and methodologies for privacy by design [20, 50]. As a first step in this direction, we focus in this paper on the data minimization principle which stipulates that the collection should be limited to the pieces of data strictly necessary for the purpose, and we provide a framework to reason about the choices of architecture and their impact in terms of privacy. The first strategic choices are the allocation of the computation tasks to the nodes of the architecture and the types of communications between the nodes. For example, data can be encrypted or hashed, either to protect their confidentiality or to provide guarantees with respect to their correctness or origin. The main benefit of a centralized architecture for the "central" actor is that he can trust the result because he keeps full control over its computation [42]. As we can see from the examples in Section 2, the loss of control by a single actor in decentralized architectures can be offset by extra requirements ensuring that errors (or frauds) can be detected *a posteriori*.

In this paper, we focus on the investigation of the architectural choices based on these criteria, especially the decentralization and error detection requirements. In order to help the designer grasp the combination of possible options, we propose a framework to express the parameters to be taken into account (the service to be performed, the actors involved, their respective requirements, etc.) and an inference system to derive properties such as the possibility for an actor to detect potential errors (or frauds) in the computation of a variable. This inference

[1] http://ec.europa.eu/justice/newsroom/data-protection/news/120125-en.htm

system can be used in the design phase to check if an architecture meets the requirements of the parties or to point out conflicting requirements.

The rest of the paper is organized as follows. Section 2 presents, as a motivating example, different architectural choices for electronic traffic pricing systems. Section 3 introduces our formal framework and establishes the correctness of our inference system. This framework is applied to the motivating example in Section 4. Section 5 discusses related work. Section 6 concludes the paper and outlines directions for further work.

2 Motivating example

Electronic Traffic Pricing (ETP) makes it possible to replace flat road tax schemes by systems in which the fee to be paid by the drivers depends on a variety of parameters related to their actual usage of the roads such as, for example, the type of roads they have used, the time of use, the traffic conditions, weather conditions, etc. These systems can be used to provide incentives for drivers to avoid using congested roads during peak hours and thus contributing to reduce traffic jams and pollution. For this reason, there are more and more initiatives around the world to deploy this kind of system on sections of roads, in urban areas, or even at the level of entire countries such as the Netherlands. These systems have obvious benefits, but they can also represent new risks for privacy. In this section, we review different architectural choices for ETP and analyze their impact in terms of privacy.

2.1 First option (centralized)

The first and maybe most natural option is the centralized architecture. This option relies on the idea that the on board equipment (OBE) of each vehicle includes a device to get its geographical position (e.g. GPS) and communication means (e.g. GSM) to send all location data to the server of the traffic pricing authority. The server computes the fee due for each car and the authority periodically sends the bill to the driver (e.g. every quarter). In addition, in order to make it possible to detect potential misbehaviors from the drivers (e.g. drivers tampering with their GPS device or turning it off), the authority is allowed to perform sporadic spot checks (similar to existing speed limitation spot checks).

This option is rather secure for the traffic pricing authority but it is highly intrusive for the drivers because the authority becomes aware of all the whereabouts of all the vehicles. This solution had originally been chosen by the Dutch government but it has trigerred a lot of protests, precisely for this reason, and the project has been suspended.

2.2 Second option (secure OBE)

A first alternative is to avoid the disclosure of any location information, which is possible if the computation of the fee can be performed by the OBE. In this case, the only data sent by the vehicles is the fee due at the end of each period (e.g. quarter). However, for this option to be acceptable for the pricing authority, the computation of the fee should be performed by a trusted device (e.g. a smart card). In addition, for the same reasons as above, the authority should be able to conduct spot checks. These spot checks are a bit more sophisticated than in the first option though, because it is necessary to communicate with the car to check that the observed location has been correctly provided as input to the OBE[2].

[2]Even if the OBE is secure, its result might be wrong if the location values provided as inputs have been tampered with.

This solution is excellent with respect to data minimization but it requires more expensive OBEs. In addition, it is necessary to provide a solution to update the fee calculation software securely (because the fees are likely to evolve during the life time of the equipment).

2.3 Third option (commitments)

A possibility to avoid the drawbacks of the previous solution is to resort to a *commitment scheme* [22]. Commitment schemes provide two key guarantees: (1) a commitment γ about a value η is such that η cannot be discovered (or "opened") by the receiver without the help of the sender and (2) it binds the sender (it is not feasible for the sender to find another value η' consistent with γ). In this solution, the OBE sends commitments to the location data to the server of the pricing authority. Like in the previous option, it performs the computation of the fee and sends it to the operator at the end of each period. The authority can initiate a verification protocol after each spot check. This protocol leads to the disclosure of partial sums of the fee to allow the authority to check that the observed position has been correctly taken into account in the computation of the global fee.

In contrast with the previous option, no secure device is required in this solution: the confidence of the authority relies on the commitment scheme and the possibility to conduct spot checks. This solution offers a high level of privacy protection to the drivers. It still leads to non minimal disclosures of data during spot checks and it can be improved using homomorphic commitments [2] (which allow for the verification of partial sums without any disclosure of the actual values).

2.4 Need for reasoned decisions

Other solutions are possible for privacy friendly ETP such as the protocol proposed in [44] based on anonymous communications and commitments to anonymous tags. The goal of this section is not to provide a comprehensive survey on privacy in ETP but to illustrate the fact that, to provide a given service, a wide variety of design choices may be available, relying on the same building blocks (commitments, spot checks and secure computation here) and leading to more or less privacy friendly solutions. One of the most important challenges for the development of privacy by design is therefore to be able to provide tools to help designers facing this combination of possibilities and to ensure they can make the best choices in terms of privacy following a rigorous and reasoned approach.

3 Formal Framework

In order to be able to reason about architectural choices, it is necessary to integrate into a single framework all the parameters that can have an impact on the architecture and its properties. The first parameter is obviously the service to be provided by the system under design. The second parameter is the set of actors involved and their respective requirements. These requirements can express the need for an actor to get access to a given information or to ensure that another actor cannot get access to the information. Other requirements are related to the possibility for an actor to challenge the provider of the information to detect potential errors ("detectability" in the sequel). Obviously, architectural choices also depend on the functionalities of the available components (e.g. encryption, commitments, secure computation, etc.) and the associated guarantees. In the following subsections, we present successively our framework for the specification of the architectures and the requirements of the actors (Subsections 3.1), the associated trace

based semantics (Subsection 3.2) and an inference system that can be used for the verification of detectability (Subsection 3.3).

3.1 Architectures

The starting point of the design phase is the identification of the set Ω of actors involved and the specification of the service to be delivered. We assume in a first stage that the service is defined as a set of equations Σ defined on variables in Var with values in Val and we write $X =_\Sigma F(Y_1, \ldots Y_2)$ an equation in Σ. $Dep(X)$ denotes the set of variables on which X depends (involved in the equations that contribute to the definition of X) and In the set of input variables (variables that do not appear in the left hand side of any equation). By convention, terminal variables are the inputs of the system provided (computed) by a specific actor called the *environment*. For example, variables representing the actual locations of the vehicles in the ETP use case are terminal variables. To illustrate our approach, we use the following set of operations (or basic blocks) in this paper :

$$Op \quad = \quad \{Compute, Send, Commit, Open, Get\}$$

Based on this set of operations, the domain of architectures is defined as follows:

$$\begin{aligned}
Arch &= (\overline{Comp} \times \overline{Send} \times \overline{Commit} \times \\
&\quad \overline{Means} \times \overline{Trusted}) \\
\overline{Comp} &= \Omega \to Vars \\
\overline{Send} &= (\Omega \times \Omega) \to Vars \\
\overline{Commit} &= (\Omega \times \Omega) \to Vars \\
\overline{Means} &= \mathcal{P}((\overline{Open} \times \overline{Get})) \\
\overline{Open} &= (\Omega \times \Omega) \to Vars \\
\overline{Get} &= \Omega \to Vars \\
\overline{Trusted} &= Vars \\
Vars &= \mathcal{P}(Var)
\end{aligned}$$

An architecture (C, S, K, M, T) defines the sets of variables which can be computed (C), sent (S), commited (K), spot checked or opened after a commitment (M), or trusted (T). A variable is trusted if the correctness of the computation of the equation defining it in Σ can be assumed. In practice, a trusted variable could be computed by a secure element such as a smart card or a secure OBE as discussed in Section 2. $Means$ is defined as a powerset because the sets of spot checked or opened variables cannot be fixed once for all, they can be chosen randomly by the actors allowed to perform these operations[3]. We consider only *consistent* architectures here, i.e. architectures such that an actor cannot both receive and compute a variable, or compute it and spot check it, etc.

We call a *context* a set of tuples made of the sets of variables which can be respectively received, opened or spot checked by an actor in a run:

$$Context = \mathcal{P}(Vars \times Vars \times Vars)$$

The function VC returns, for each architecture and actor, the associated valid context:

$$VC \quad \in \quad (Arch \times \Omega) \to Context$$

[3]In the same way as car drivers cannot generally predict if and where their speed will be checked by a radar.

$$
\begin{array}{l}
VC((C,S,K,M,T),A) = \{(R,O,G)| \\
R = \{X|\exists B \in \Omega, X \in S(B,A)\} \\
\wedge\ \exists (F_O,F_G) \in M \\
\quad O = \{X|\exists B \in \Omega, X \in F_O(B,A)\} \\
\quad G = F_G(A)\}
\end{array}
$$

The intuition behind this definition is that, for a given architecture $\mathcal{A}$ and actor A, the operations available to A for collecting information is characterized by $VC(\mathcal{A}, A)$. In other words, each run of A must be covered by an element (R, O, G) of $VC(\mathcal{A}, A)$: all variables received, opened or spot checked by A should belong respectively to R, O and G. The role of the function VC is therefore to extract, for a given actor, the set of operations (context) authorized by the architecture.

A first way to express the requirements of the actors is to use constraints on the components of the architecture. For example, if (C, S, K, M, T) denotes the architecture, the fact that an actor A does not want to disclose the value of a variable X to an actor B, neither in clear nor as a commitment, can be expressed as $X \notin S(A,B) \cup K(A,B)$. If A wants to restrict spot checks to variables $\{X_1, \ldots, X_n\}$ and to limit them to 1 in a run, the constraint can be expressed as $\forall (F_O, F_G) \in M, F_G(B) \subseteq \{X_1, \ldots, X_n\} \wedge Card(F_G(B)) \leq 1$. Constraints on the side of the controller can be expressed in the same way. For example, the fact that A wants to compute himself the value of X is expressed as $X \in C(A)$. We show in the following sections how to express detectability properties.

3.2 Semantics

In order to be able to reason about the correctness of an architecture with respect to detectability requirements, it is necessary to define the semantics of the operations. We first define the domains of global traces and local traces (respectively $\overline{\Theta}$ and Θ).

$$
\begin{array}{rcl}
\overline{\Theta} & = & \Omega \rightarrow \Theta \\
\Theta & = & Seq(Event) \\
Event & = & \{O_A^B(X,V)|A \in \Omega, B \in \Omega, X \in Var, \\
 & & V \in Val, O \in \{Send, Commit, Open\}\} \\
 & & \cup \{O_A(X,V)|A \in \Omega, X \in Var, V \in Val, \\
 & & O \in \{Compute, Get\}\}
\end{array}
$$

A local trace is a *consistent* sequence of events associated with a given actor, each event corresponding to the occurrence of an operation involving this actor. $Compute_A(X,V)$, $Get_A(X,V)$, $Send_A^B(X,V)$, $Commit_A^B(X,V)$ and $Open_A^B(X,V)$ are events denoting respectively : the computation of variable X by A (resulting in the value V), the spot check of variable X by A, the communication of the value V of the variable X from B to A, the commitment of B (towards A) to the value V of X, and the opening of the commitment by A. Note that the value V which appears in $Commit_A^B(X,V)$ is not revealed to A; in contrast, $Open_A^B(X,V)$ discloses to A a value V to which B is commited (i.e. for which B has previously issued a $Commit_A^B(X,V)$ event).

We consider only *consistent* sequences of events in this report, which are sequences of events in which variables are used consistently: a variable cannot be both computed and received by an actor, is received or computed only once, can be opened only to the value corresponding to an earlier commitment and sent only if it has been previously received or computed. The definition of consistent sequences is provided in Annex 1.

It should be noted that focusing on consistent sequences is not a restriction of the framework. For example, executions in which an opened value is not consistent with an earlier commitment can be immediately detected by the actor performing the opening. Similarly, receiving a value which has already been computed or received would be a breach of the architecture detected by the receiver. As far as the threat model is concerned, any tampering with the variables or malicious action from an actor can be expressed through the *Compute* operations which are completely unrestricted: the value V in a $Compute_A(X, V)$ event does not need to satisfy the equation defining X in Σ. As shown in the case study in Section 4, it is possible to introduce as many intermediate variables as necessary in the set of equations Σ to account for all potential threats.

The state of an actor is defined as a function in

$$St \quad = \quad Var \rightarrow Val_\perp$$

where $Val_\perp$ is the domain of values extended with $\perp$, which denotes the undefined value, and values of type "Commitment", which are written symbolically as $\Xi(V)$. As explained above, $\Xi(V)$ does not provide any information about V itself. By abuse of notation, we also write $\perp$ the error state.

Definition 1 The state of an actor A after the execution of the operations in a trace σ is defined by $S_A(\sigma, \eta_0)$, where η_0 stands for the initial state. By abuse of notation, we write $S_A(\sigma)$ for $S_A(\sigma, \emptyset)$ in the sequel, with $\emptyset$ the empty environment.

$$\begin{aligned} S_A(\langle\rangle, \eta) &= \eta \\ S_A(e.\sigma, \eta) &= S_A(\sigma, T_A(e, \eta)) \end{aligned}$$

$$\begin{aligned} T_A(Compute_A(X,V), \eta) &= \eta[V/X] \\ T_A(Compute_B(X,V), \eta) &= \eta \ \ if\ A \neq B \\ T_A(Send_A^B(X,V), \eta) &= \eta[V/X] \\ T_A(Send_B^A(X,V), \eta) &= \eta \ \ if\ A \neq B \\ T_A(Commit_A^B(X,V), \eta) &= \eta[\Xi(V)/X] \\ T_A(Commit_B^A(X,V), \eta) &= \eta \ \ if\ A \neq B \\ T_A(Open_A^B(X,V), \eta) &= \eta[V/X]\ if\ \eta(X) = \Xi(V) \\ &= \perp\ otherwise \\ T_A(Open_B^A(X,V), \eta) &= \eta \ \ if\ A \neq B \\ T_A(Get_A(X,V), \eta) &= \eta[V/X] \end{aligned}$$

The expression $\eta[V/X]$ denotes a state similar to η except that V is bound to X. The only operations that have an impact on the state of A are $Compute_A(X, V)$, $Send_A^B(X, V)$, $Commit_A^B(X, V)$, $Open_A^B(X, V)$ and $Get_A(X, V)$. Note that the value discovered through an *Open* operation must be consistent with a commitment received previously ($\eta(X) = \Xi(V)$); otherwise the resulting state is $\perp$.

3.3 Detectability

The requirements on the possibility (or impossibility) for an actor to have access to a given variable (either by default, or sporadically through a spot check or the opening of a commitment)

can be verified by simple static reasoning (akin to flow analysis) based on the definition of the architecture. The situation is much more complex for the possibility to detect a potential error (or fraud) in the computation of a variable. Indeed, because an actor may not control entirely the computation of a variable, he can generally not be sure that the value of this variable in his environment is correct. To be able to reason about the possibility for an actor to detect errors, we introduce an inference system and we establish its correctness with respect to the semantics defined in the previous subsection.

Definition 2 The inference system is made of the following three rules:

(R1) *If* $X =_\Sigma F(Y_1, \ldots, Y_n)$ *and* $X \in T$ *then*

$$\frac{\forall i \in \{1, \ldots, n\}, U_i \vdash_T Y_i}{\bigcup_{i=1}^{n} U_i \vdash_T X}$$

(R2) *If* $X =_\Sigma F(Y_1, \ldots, Y_n)$ *and* $X \notin T$ *and*

$$R' \cup O' \cup G' = \{X, Y_1, \ldots, Y_n\} \quad \textit{then}$$

$$\frac{\forall i \in \{1, \ldots, n\}, U_i \vdash_T Y_i}{U \vdash_T X}$$

$$\textit{with } U = \{(R \cup R', O \cup O', G \cup G') | (R, O, G) \in \bigcup_{i=1}^{n} U_i\}$$

(R3) *If* $X \in In$ *then* $U \vdash_T X$

The intuition behind these rules is as follows:

- Rule R1 corresponds to the case of a trusted variable ($X \in T$). As stated in Section 3.1, a trusted variable is a variable such that the computation of the associated equation can be assumed to be correct. Therefore it is sufficient to be able to check the variables $Y_1, \ldots, Y_n$ used to compute X.
- The motivation for rule R2 is similar except that X cannot be trusted. Therefore, it must be possible in addition to check that the equation $X =_\Sigma F(Y_1, \ldots, Y_n)$ itself is satisfied. To this aim, it is necessary to be able to collect the values of all variables involved, hence the definitions of R', O', G' and U: the variables $X, Y_1, \ldots, Y_n$ are split up into the sets R', O' and G' which are added to the respective sets R, O and G in the context of the conclusion of the rule.
- Rule R3 deals with terminal variables (elements of In as introduced in Subsection 3.1). By assumption, these variables are the genuine values provided by the environment. Therefore they can be trusted in any context.

Before stating the correctness of this inference system, we need to introduce some properties on traces.

Definition 3 If $A \in \Omega$, $\sigma \in \Theta$, $U \in Context$, $Complete_A(U, \sigma)$ holds if and only if $\forall (R, O, G) \in U$,

$$\forall X \in R, \exists V \in Val, \exists B \in \Omega, Send_A^B(X, V) \in \sigma \quad \textit{and}$$
$$\forall X \in O, \exists V \in Val, \exists B \in \Omega, Commit_A^B(X, V) \in \sigma$$

By abuse of notation, we use the symbol $\in$ to denote membership to a sequence. A trace σ is complete with respect to a context U if all the variables that can be sent or committed have actually been sent or committed. Hence, completeness here means that all communications which are at the initiative of the sender have occured.

Definition 4 If $\sigma \in \Theta$ and $X \in Var$, $Incorrect\ (\sigma, X)$ holds if and only if

$$S(\sigma)(X) \neq \perp \quad \wedge \quad \forall V \in Val, S(\sigma)(X) \neq \Xi(V) \quad \wedge \quad S(\sigma)(X) \neq Eval(X, S(\sigma))$$

A variable X is incorrect after the execution of a trace σ if it has a value that is different from the value derived from the equations in Σ. Note that the incorrectness of a variable X may remain unknown from the actor A after the execution of a trace σ (because A may not know the values of the terminal variables). $Eval(X, \eta)$ defines the correct value of a variable X assuming that the input variables are defined by η:

Definition 5 If $X \in Var$ and $\eta \in St$,
$Eval(X, \eta) = if\ X =_\Sigma F(Y_1, \ldots, Y_n)$
$\quad then\ \ F(Eval(Y_1, \eta), \ldots, Eval(X_n, \eta))$
$\quad else\ \eta(X)$

The trust assumption associated with the T component of an architecture can now be defined in terms of the $Incorrect$ relation:

Definition 6 $\forall \sigma \in \Theta$, $\forall (C, S, K, M, T) \in Arch$, $\forall X \in T$,
$X =_\Sigma F(Y_1, \ldots, Y_n)$ and $Incorrect(\sigma, X) \Rightarrow \exists j \in [1, n], Incorrect(\sigma, Y_j)$.

The definition expresses the fact that if the value V of a trusted variable X defined by $X =_\Sigma F(Y_1, \ldots, Y_n)$ is incorrect, it must be the case that the value of one of the variables Y_j is incorrect because the computation of the equation associated with X itself must be correct.

We can now introduce the function $Detect$ which defines what we mean by "being able to detect any error in the computation of variable".

Definition 7 If $A \in \Omega, \sigma \in \Theta$, $X \in Var$ and $U \in Context$, $Detect_A^U(\sigma, X)$ holds if and only if

$$\begin{array}{l}
\exists Z \in Dep(X), Z =_\Sigma F(Z_1, \ldots, Z_n), \exists (R, O, G) \in U, \\
\exists Y_1, \ldots, Y_m \in G, \exists Y'_1, \ldots, Y'_{m'} \in\ O, \\
\quad E(Y_1) = \ldots = E(Y_m) = \perp \\
\quad E(Y'_1) = \Xi(V'_1), \ldots, E(Y'_{m'}) = \Xi(V'_{m'}) \\
\quad \wedge\ \sigma' = \sigma.\ Get(Y_1, V_1) \ldots Get(Y_m, V_m). \\
\qquad\qquad Open(Y'_1, V'_1) \ldots Open(Y'_{m'}, V'_{m'}) \\
\quad \wedge E'(Z) \neq F(E'(Z_1), \ldots, E'(Z_n)) \\
with\ E = S_A(\sigma)\ and\ E' = S_A(\sigma')
\end{array}$$

The intuition behind the definition of $Detect_A^U(\sigma, X)$ is that, in order to be able to detect an error in the value of X, A should be able to apply Get and $Open$ operations allowed in the context U (definition of σ') to reach a state E' in which he has the proof of an inconsistency in the values of variables $Z, Z_1, \ldots, Z_n$. This inconsistency $(E'(Z) \neq F(E'(Z_1), \ldots, E'(Z_n)))$ reveals an error in the value of X itself because $Z \in Dep(X)$ (which means that the value of X depends on the value of Z).

We can now state the correctness property of our inference system:

Theorem 1 $\forall A \in \Omega, \forall X \in Var, \forall U \in Context, \forall T \in Vars,$

$$\begin{array}{l} if\ U \vdash_T X\ \ then\ \ \forall \sigma \in \Theta \\ Complete_A(U, \sigma) \wedge Incorrect(\sigma, X) \Rightarrow \\ Detect_A^U(\sigma, X) \end{array}$$

The correctness property states that if U is a valid context for an actor A in an architecture (C, S, K, M, T) and we can derive $U \vdash_T X$, then it must be the case that A can detect any error in the computation of X. In other words, any complete trace can be extended by A into a trace leading to an inconsistent state (with respect to the equations of Σ).

The correctness of Theorem 1 can be proved by induction on the derivation tree of statements $U \vdash_T X$ considering each rule in turn. The proof is provided in Annex 1.

4 Application

In this section, we illustrate the framework presented in the previous section with the ETP case study described in Section 2. In order to instantiate the framework to a given application, we need to define the service to be delivered, the set of actors involved and their respective requirements. We assume that the ETP service is the computation of the fee due by each driver for a given period of time, for example a quarter. This service can be defined by the following system of equations Σ:

$$\begin{array}{rcl} Q & =_\Sigma & M_1 + M_2 + M_3 \\ M_i & =_\Sigma & D_{i,1} + \ldots D_{i,31} \\ D_{i,j} & =_\Sigma & H_{i,j,1} + \ldots H_{i,j,144} \\ H_{i,j,k} & =_\Sigma & F(P_{i,j,k}) \\ P_{i,j,k} & =_\Sigma & A_{i,j,k} \end{array}$$

Variables $Q, M_i, D_{i,i}, H_{i,j,k}$ represent the fees due for, respectively, a quarter, a month, a day and a ten minutes slot. $P_{i,j,k}$ are the position variables used to compute the fees whereas $A_{i,j,k}$ denote the actual (genuine) positions of the vehicle. In general, it is useful to distinguish different occurrences of variables to account for potential discrepencies resulting from frauds or errors occuring during their communication. It is necessary to distinguish between $A_{i,j,k}$ and $P_{i,j,k}$ here because the former is an environment variable whereas the latter is under the control of the OBE. Similar distinctions could have been introduced between the other variables to account for potential communication errors, but we choose to limit the number of variables for the sake of conciseness.

We consider three actors here, namely the driver δ, the pricing authority α and the environment ϵ: $\Omega = \{\delta, \alpha, \epsilon\}$. Let us denote by $\mathcal{A} = (C, S, K, M, T)$ the architecture to be defined. The first requirement of the pricing authority, which is to be met in all scenarios, is that it should be able to detect any error in the computation of the fee Q:

$$VC(\mathcal{A}, \alpha) \vdash_T Q$$

A first scenario corresponds to the additional requirement that the pricing authority receives all position data $P_{i,j,k}$ and performs all computations.

$$P_{i,j,k} \in S(\delta, \alpha) \wedge \{H_{i,j,k}, D_{i,j}, M_i, Q\} \quad \in \quad C(\alpha)$$

By convention, non quantified indexes are implictly quantified over their respective ranges ($i \in [1..3], j \in [1,31], k \in [1,144]$). There are several cases in which these requirements can conflict with the requirements of the driver: the first and obvious case is when the driver does not accept to disclose any location data ($\forall i,j,k, P_{i,j,k} \notin S(\delta,\alpha)$) or not all of them. But it is also the case if the driver does not accept spot checks ($\forall (O,G) \in M, G = \emptyset$) because it is then impossible to establish $VC(\mathcal{A},\alpha) \vdash_T Q$. The reason is that the authority is not able to detect an error in the last equation of Σ: $P_{i,j,k} = A_{i,j,k}$. If this requirement is relaxed, for example into $\forall (O,G) \in M, G \subseteq \{A_{i,j,k}\} \wedge Card(G) \leq 1$, which allows for one single spot check in a quarter, the conflict disappears and the inference system allows us to prove $VC(\mathcal{A}_1,\alpha) \vdash_{T_1} Q$ with $\mathcal{A}_1 = (C_1, S_1, K_1, M_1, T_1)$ defined as follows:

$$\begin{aligned} C_1(\alpha) &= \{Q, M_i, D_{i,j}, H_{i,j,k}\} \\ S_1(\delta,\alpha) &= \{P_{i,j,k}\} \\ M_1 &= \{(O, G_{i,j,k}) | G_{i,j,k}(\alpha) = \{A_{i,j,k}\}\} \\ T_1 &= \{Q, M_i, D_{i,j}, H_{i,j,k}\} \end{aligned}$$

By convention, all other sets are empty. This architecture corresponds to the first option of Section 2 (centralized solution) and it is easy to check that $U_1 \vdash_{T_1} Q$ with $U_1 = VC(\mathcal{A}_1,\alpha)$ can be derived through the application of rules R2 and R3 (Subsection 3.3) on the system of equations Σ defining Q.

Let us consider now another scenario in which the driver does not accept that any location data is disclosed to the pricing authority but the OBE is equipped with a secure component for the computation of the fee. If the driver imposes the same constraint as above on spot checks, we get the architecture $\mathcal{A}_2 = (C_2, S_2, K_2, M_2, T_2)$ defined as follows:

$$\begin{aligned} C_2(\delta) &= \{Q, M_i, D_{i,j}, H_{i,j,k}\} \\ S_2(\delta,\alpha) &= \{Q\} \\ M_2 &= \{(O, G_{i,j,k}) | G_{i,j,k}(\alpha) = \{A_{i,j,k}\}\} \\ T_2 &= \{Q, M_i, D_{i,j}, H_{i,j,k}\} \end{aligned}$$

However, this architecture does not satisfy $VC(\mathcal{A}_2,\alpha) \vdash_{T_2} Q$ because it is impossible to prove $U \vdash_{T_2} P_{i,j,k}$ for any valid context U. The reason is that, even if it can conduct spot checks, the authority does not have any means to detect that the genuine location data have been provided as inputs to the secure component (that is to say that $P_{i,j,k} = A_{i,j,k}$). To check this, the authority must be able, after a spot check of a position $A_{i,j,k}$, to get also the corresponding value $P_{i,j,k}$. This observation leads to the architecture $\mathcal{A'}_2 = (C'_2, S'_2, K'_2, M'_2, T'_2)$ which satisfies $VC(\mathcal{A'}_2,\alpha) \vdash_{T'_2} Q$:

$$\begin{aligned} C'_2(\delta) &= \{Q, M_i, D_{i,j}, H_{i,j,k}\} \\ S'_2(\delta,\alpha) &= \{Q\} \\ M'_2 &= \{(O, G_{i,j,k}) | G_{i,j,k}(\alpha) = \{P_{i,j,k}, A_{i,j,k}\}\} \\ T'_2 &= \{Q, M_i, D_{i,j}, H_{i,j,k}\} \end{aligned}$$

The proof of $U'_2 \vdash_{T'_2} Q$ with $U'_2 = VC(\mathcal{A'}_2,\alpha)$ can be derived by the application of rule R3 to get $U'_2 \vdash_{T'_2} A_{i,j,k}$, followed by the application of rule R2 to prove $U'_2 \vdash_{T'_2} P_{i,j,k}$ and the application of rule R1 to derive successively $U'_2 \vdash_{T'_2} H_{i,j,k}$, $U'_2 \vdash_{T'_2} D_{i,j}$, $U'_2 \vdash_{T'_2} M_i$ and $U'_2 \vdash_{T'_2} Q$.

The above architecture corresponds to option 2 in Section 2. The third option, which is based

on commitments[4], can be defined as $\mathcal{A}_3 = (C_3, S_3, K_3, M_3, T_3)$ with:

$$\begin{aligned} C_3(\delta) &= \{Q, M_i, D_{i,j}, H_{i,j,k}\} \\ S_3(\delta, \alpha) &= \{Q\} \\ M_3 &= \{(O_{i,j,k}, G_{i,j,k}) | O_{i,j,k}(\delta, \alpha) = \\ &\quad \{P_{i,j,k}, H_{i,j,k'}, D_{i,j'}, M_{i'} | \\ &\quad i' \in [1..3], j' \in [1, 31], k' \in [1, 144]\} \wedge \\ &\quad G_{i,j,k}(\alpha) = \{A_{i,j,k}\}\} \end{aligned}$$

In this case, the only information that the authority can get by spot checks is, as in $\mathcal{A}_1$, one position of the vehicle ($\{A_{i,j,k}\}$) per quarter. As in $\mathcal{A}'_2$, all computations are done by the OBE. However, in contrast with $\mathcal{A}'_2$, no trust assumption is made here ($T_3 = \emptyset$). Detectability comes from the possibility to disclose commitments after a spot check: the intuition behind $O_{i,j,k}(\delta, \alpha)$ is that all variables contributing to the computation of the part of the fee in which $\{A_{i,j,k}\}$ is involved must be disclosed for the authority to be able to check that they have been correctly included in the computation of the quarterly fee Q.

5 Related Work

Privacy by design has been strongly advocated by the Information and Privacy Commissioner of Ontario [8, 9] and it has been praised by a number of academic lawyers as an essential step towards a better privacy protection [45].

On the technological front, privacy enhancing technologies (PETs) have been an active research topic in computer science during the last decades [15, 19, 46] and a variety of techniques have been proposed (including anonymizers, identity management systems, privacy proxies, encryption mechanisms, filters, anonymous credentials, commitment schemes, sanitization techniques, etc.). As discussed in Section 1 and Section 2, these techniques have been applied in a variety of areas[5], but on a case by case basis and "privacy by design" is generally not addressed from a general perspective. As pointed out in [20], privacy by design "requires the development of generalizable methodologies that build upon the principle of data minimization". The goal of this report is precisely to propose a formal framework to address this need.

As far as formal models for privacy are concerned, previous work in this area can be classified into three main categories:

- *Language based approaches:* a number of languages and logics have been proposed to express privacy policies [1, 3, 4, 5, 11, 12, 13, 21, 23, 29, 24, 34, 37, 52]. These languages may target citizens, businesses or organizations; they can be used to express individual privacy policies, corporate rules or legal rules. Not all of them are endowed with a formal semantics though. When it is the case (e.g. [1, 3, 4, 5, 29, 21, 34, 37, 52]), they can be used to verify consistency properties or to check if a system complies with a privacy policy. These verifications can be performed either *a priori*, through static verification techniques, on the fly, using monitoring, or *a posteriori* in the context of audits or accountability procedures. The policies expressed in these languages are usually more fine-grained than the properties considered here and they tend to be more complete with respect to privacy

[4]For the sake of the example, we consider the version with commitment trees here, i.e. commitments on location data and on partial sums.

[5]For example, ubiquitous systems in [28], smart metering in [18, 32, 47], pay as you drive in [22, 2, 44], or location privacy in [14, 25, 26].

(e.g. including notions of obligations or data deletion). In contrast with the framework proposed here, they do not provide ways to reason about architectural choices, in particular about the relationship between trust requirements and decentralization, which is the heart of this report.

- *Decentralized security models*: the decentralized label model [41] makes it possible to reason about information flows between principals that do not trust each other. Labels are used to express confidentiality requirements on the data. They define, for each principal, the authorized readers of their data. The model can be extended to include the authorized writers in order to express integrity constraints. The decentralized label model has been applied to the Jif programming language [41]: labels can be associated with variables and checked by static analysis. They can also be used to split Jif programs securely, i.e. to derive a distributed implementation that statisfies all policies of the principals [53]. Labels have also been used in Fabric, another extension of Jif with support for secure distributed programming [35]. The decentralized label model could be used to express certain aspects of our framework, such as the $\overline{Send}$ component of our architectures, but it is not suitable to reason about detectability properties.

- *Privacy metrics*: notions such as k-anonymity [33, 48], l-diversity [36] or ϵ-differential privacy [16, 17] have been proposed as ways to measure the level of privacy provided by an algorithm. Methods [17, 39, 38] have been proposed to design algorithms achieving these privacy metrics (e.g. through the deletion of values, generalization or the introduction of noise) or to verify that a system achieves a given level of privacy [49]. These contributions on privacy metrics are complementary to the work described in this report. We have followed a logical (or qualitative) approach here, proving that a given privacy property is met (or not) by an architecture. As suggested in the next section, an avenue for further research would be to cope with quantitative reasoning as well, using inference systems to derive properties expressed in terms of privacy metrics.

6 Discussion and further work

The framework presented in this report has been applied to the verification of architectures for electronic traffic payment systems and smart metering. The tool, which is implemented in Haskell, provides different modes of use: fully automatic, assisted and manual. In all cases, the first task of the user is to define the fixed parameters of the problem, as specified in Section 3 (the service defined as a set of equations and the requirements of the actors defined as constraints on sets of variables). In the fully automatic mode, the user just requests the proof of detectability of a given variable X for an actor A. The tool then looks for all architectures meeting the constraints that can lead to a proof of X by the inference system defined in Section 3. In the assisted mode, the same principle applies but the search is limited to the application of a rule (R1, R2 or R3) proposed by the user. In the manual mode, the user provides not only the rule to apply but also the variable to check. The rationale for the use of these modes is the following: when the designer is able to define sufficiently constraining requirements (either because he already has a good intuition about the appropriate architecture or because the constraints imposed by the actors are strong enough), he can use the fully automatic mode to confirm his intuition or to check that the constraints can be met. The other modes can be used either if the initial constraints are too loose[6] or in "debugging mode" to understand why the system fails to find a

[6]The response time of the automatic mode can become unacceptable in such cases because the worst case complexity of the search (when no constraint at all are imposed on the architecture) is exponential in terms of the

proof for a given variable. In general, the designer should strive to express from the start all obvious constraints or to use in a first stage the tool in assisted mode to better understand the options investigated by the system and possibly refine iteratively his initial set of constraints.

We would like to emphasize that several conditions have to be met for the above tool and the overall approach to be applicable:

- First, it must be possible to define the service to be provided by the system as the result of a computation involving the input data (e.g. the computation of a fee in ETP or smart metering, or the computation of a test to decide whether a given ad should be sent to an internet user). This definition plays a pivotal rôle in the analysis of acceptable architectures. Thus the approach does not help in situations such as social networks where the service is just the display of the data (and its access based on a given privacy policy).

- In addition, the framework proposed here does not provide off-the-shelf solutions nor answers to broad questions such as: "What is the best architecture to solve this problem?". It provides answers to specific questions such as "Given this service to be delivered, this set of constraints from the actors involved and this set of available operations (building blocks), what are the acceptable architectures or is this architecture acceptable?". In addition, it is necessary to be able to provide a formal characterization of all the aforementioned parameters. Section 3 presents such a formalization for ETP systems, given a set of available operations and we suggest below how other operations could be dealt with.

- In this report, we have considered only one of the privacy by design principles, namely data minimization; other principles such as, for example, transparency or accountability [51] are also of upmost importance and require further research.

Beyond this framework, the goal of this report is to put forward a formal approach to privacy by design [31] which can be used as a foundation for a systematic exploration of the design space and the justification of architectural choices. A systematic method is needed for at least two reasons: first, privacy is a very complex issue, which may sometimes conflict (or seem to conflict) with other requirements. Secondly, a wide variety of Privacy Enhancing Technologies (PETs) are available and many more will be proposed in the future. Tools are thus badly needed to allow designers to master this complexity and to take decisions based on rigorous grounds. Before giving up on privacy on the pretext of apparently conflicting requirements, all options must be considered and the controller should be in a position to prove that no other solution can meet the functional constraints while collecting less personal data. This requirement is also in line with the accountability principle of the draft regulation published by the European Commission in January 2012[7].

The framework introduced in Section 3 is a first step in this direction and an illustration of a more general approach. For example, the definition of architectures is based on the set of operations or buiding blocks available. In this report, we have chosen a set of operations useful to investigate a class of solutions for ETP. The same set of operations can be applied to other application areas such as smart metering. But other techniques can be included as well in the set of operations and the corresponding sets added to architectures and contexts. The condition to be able to integrate a new operation in the framework is to be able to express the relevant properties as inference rules. For example, in order to include homomorphic commitments into the framework, we need to extend architectures with an additional set K_F (for variables that are

number of variables involved in the definition of the services (because the system may have to explore all possible contexts).

[7] http://ec.europa.eu/justice/newsroom/data-protection/news/120125-en.htm, Art. 22

commited using an homomorphic encryption algorithm for operation F), extend valid contexts to include this set K_F and add the following rule:

(R4) If $X =_\Sigma F(Y_1, \ldots, Y_n)$, $X \notin T$ and $\{X, Y_1, \ldots, Y_n\} \subseteq K_F$

$$\frac{\forall i \in \{1, \ldots, n\}, U_i \vdash_T Y_i}{\bigcup_{i=1}^{n} U_i \vdash_T X}$$

This rule expresses the fact that if all the variables involved are commited using an homomorphic scheme, the only requirement is to be able to detect an error in the computation of the variables $Y_1, \ldots, Y_n$ because the validity of the equation $X =_\Sigma F(Y_1, \ldots, Y_n)$ can be checked directly on the commitments.

Other straightforward enhancements to the formalism are possible, at the price of extra administration burden, such as the distinction between nodes and actors or the introduction of constraints on the physical architecture (such as, for example, the possibility to implement a given computation on a given node or to have a communication link between two nodes). It would also be possible to consider a richer input language to express services and to provide an abstraction function to extract (from specifications in this richer language) the equations used here to express the dependencies between variables.

A complementary extension would be the introduction of an inference system to reason about the knowledge of the actors in a more abstract way than done in Section 3. In this report, we decided to focus on the detectability constraint because it has received less attention so far, but the possibility for an actor to know a given information (beyond the fact that he may receive or not a given set of variables) is obviously at the heart of privacy protection. Inference systems to prove knowledge properties (akin to epistemic logic [6, 10, 40]) can be defined independently of the detectability system introduced in Section 3 and used to check additional constraints. Such an inference system typically includes rules to prove that, from a given set of knowledge, an actor can derive a new knowledge.

Another interesting problem for the future would be the analysis of the specification of the service itself. In this report, we have taken this specification for granted and considered that all variables involved in a definition were really necessary. There may be cases where this assumption does not hold though, and it would be interesting to be able to detect this situation and to transform the initial specification into an equivalent, but less "personal data consuming" solution. This kind of analysis is reminiscent of *strictness analysis* in functional languages [27] and inspiration can be taken in this area.

Last but not least, in this report, we have followed a "logical" (or qualitative) approach, as opposed to a quantitative approach to privacy. An avenue for further research in this area would be to study the integration of quantitative measures of privacy (such as differential privacy) into the framework.

7 Acknowledgments

The author would like to thank the reviewers of the CODASPY 2013 conference for their valuable comments and suggestions to improve the document. The final version of this report has benefited substantially from their comments. Many thanks are due also to Gustavo Grieco for his implementation of the tool sketched in this report.

References

[1] M. Backes, M. Dürmuth, and G. Karjoth. Unification in privacy policy evaluation - translating EPAL into Prolog. In *POLICY*, pages 185–188, 2004.

[2] J. Balasch, A. Rial, C. Troncoso, B. Preneel, I. Verbauwhede, and C. Geuens. PrETP: Privacy-preserving electronic toll pricing. In *USENIX Security Symposium*, pages 63–78, 2010.

[3] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and contextual integrity: Framework and applications. In *IEEE Symposium on Security and Privacy*, pages 184–198, 2006.

[4] A. Barth, J. C. Mitchell, A. Datta, and S. Sundaram. Privacy and utility in business processes. In *CSF*, pages 279–294, 2007.

[5] M. Y. Becker, A. Malkis, and L. Bussard. A practical generic privacy language. In *ICISS*, pages 125–139, 2010.

[6] M. Burrows, M. Abadi, and R. M. Needham. A logic of authentication. *ACM Trans. Comput. Syst.*, 8(1):18–36, 1990.

[7] L. Bygrave. Privacy-enhancing technologies - caught between a rock and the hard place. *Privacy Law and Policy Reporter*, 9, 2002.

[8] A. Cavoukian. *Privacy and radical pragmatism: change the paradigm.* White Paper, Information and Privacy Commissioner of Ontario, Canada, 2008.

[9] A. Cavoukian. *Privacy by design. The 7 foundational principles.* White Paper, Information and Privacy Commissioner of Ontario, Canada, 2009.

[10] R. Chadha, S. Delaune, and S. Kremer. Epistemic logic for the applied pi calculus. In *FMOODS/FORTE*, pages 182–197, 2009.

[11] O. Chowdhury, H. Chen, J. Niu, N. Li, and E. Bertino. On XACML's adequacy to specify and to enforce HIPAA. In *USENIX Workshop on Health Security and Privacy*, 2012.

[12] L. Cranor, B. Dobbs, S. Egelman, G. Hogben, J. Humphrey, M. Langheinrich, M. Marchiori, M. Presler-Marshall, J. Reagle, M. Schunter, D. A. Stampley, and R. Wenning. *The Platform for Privacy Preferences 1.1 (P3P1.1) Specification.* W3C, 2006.

[13] L. Cranor, M. Langheinrich, and M. Marchiori. *A P3P Preference Exchange Language 1.0 (APPEL1.0).* W3C, 2002.

[14] M. L. Damiani, E. Bertino, and C. Silvestri. The probe framework for the personalized cloaking of private locations. *Transactions on Data Privacy*, 3(2):123–148, 2010.

[15] Y. Deswarte and C. A. Melchor. Current and future privacy enhancing technologies for the internet. *Annals of Telecommunications*, 61(3):399–417, 2006.

[16] C. Dwork. Differential privacy. In *ICALP (2)*, pages 1–12, 2006.

[17] C. Dwork. A firm foundation for private data analysis. *Commun. ACM*, 54(1):86–95, 2011.

[18] F. D. Garcia and B. Jacobs. Privacy-friendly energy-metering via homomorphic encryption. In *STM'10 Proceedings of the 6th international conference on Security and trust management*, pages 226–238. Springer, 2010.

[19] I. Goldberg. Privacy-enhancing technologies for the internet iii: ten years later. In *Digital Privacy: Theory, Technologies, and Practices*, pages 84–89. TeX Users Group, December 2007.

[20] S. Gürses, C. Troncoso, and C. Diaz. Engineering privacy by design. In *Conference on Computers, Privacy and Data Protection (CPDP 2011)*, 2011.

[21] M. Jafari, P. W. L. Fong, R. Safavi-Naini, K. Barker, and N. P. Sheppard. Towards defining semantic foundations for purpose-based privacy policies. In *CODASPY*, pages 213–224, 2011.

[22] W. D. Jonge and B. Jacobs. Privacy-friendly electronic traffic pricing via commits. In *Proceedings of the Workshop of Formal Aspects of Securiy and Trust*, pages 132–137. Springer, LNCS 5491, 2009.

[23] G. Karjoth, M. Schunter, and E. V. Herreweghen. Translating privacy practices into privacy promises -how to promise what you can keep. In *POLICY*, pages 135–146, 2003.

[24] G. Karjoth, M. Schunter, E. V. Herreweghen, and M. Waidner. Amending P3P for clearer privacy promises. In *DEXA Workshops*, pages 445–449, 2003.

[25] E. Kosta, J. Zibuschka, T. Scherner, and J. Dumortier. Legal considerations on privacy-enhancing location based services using PRIME technology. *Computer Law and Security Report*, 4:139–146, 2008.

[26] J. Krumm. A survey of computational location privacy. *Pers Ubiquit Comput*, 13:391–399, 2008.

[27] T.-M. Kuo and P. Mishra. Strictness analysis: A new perspective based on type inference. In *FPCA*, pages 260–272, 1989.

[28] M. Langheinrich. Privacy by design - principles of privacy aware ubiquitous systems. In *Proceedings of the Ubicomp Conference*, pages 273–291. Springer, LNCS 2201, 2001.

[29] D. Le Métayer. A formal privacy management framework. In *FAST (Formal Aspects of Security and Trust)*, pages 161–176. Springer, LNCS 5491, 2009.

[30] D. Le Métayer. Privacy by design: a matter of choice. In *Data Protection in a Profiled World*, pages 323–334. Springer, 2010.

[31] D. Le Métayer. Formal methods a link between software code and legal rules. In *SEFM (Software Engineering and Formal Methods)*, pages 3–18. Springer, LNCS 7041, 2011.

[32] M. LeMay, G. Gross, C. A. Gunter, and S. Garg. Unified architecture for large-scale attested metering. In *HICSS*, page 115, 2007.

[33] N. Li, W. H. Qardaji, and D. Su. Provably private data anonymization: Or, k-anonymity meets differential privacy. *CoRR*, abs/1101.2604, 2011.

[34] N. Li, T. Yu, and A. I. Antón. A semantics based approach to privacy languages. *Comput. Syst. Sci. Eng.*, 21(5), 2006.

[35] J. Liu, M. D. George, K. Vikram, X. Qi, L. Waye, and A. C. Myers. Fabric: a platform for secure distributed computation and storage. In *SOSP*, pages 321–334, 2009.

[36] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, page 24, 2006.

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

[38] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53(9):89–97, 2010.

[39] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103, 2007.

[40] J.-J. C. Meyer and W. van der Hoek. *Epistemic Logic for Computer Science and Artificial Intelligence.*

[41] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Trans. Softw. Eng. Methodol.*, 9(4):410–442, 2000.

[42] A. Narayanan, V. Toubiana, S. Barocas, H. Nissenbaum, and D. Boneh. A critical look at decentralized personal data architectures. *CoRR*, abs/1202.4503, 2012.

[43] OECD. *OECD guidelines on the protection of privacy and transborder flows of personal data, Organization for Economic Co-operation and Development.* OECD, 1980.

[44] R. A. Popa, H. Balakrishnan, and A. J. Blumberg. Vpriv: Protecting privacy in location-based vehicular services. In *USENIX Security Symposium*, pages 335–350, 2009.

[45] Y. Poullet. About the e-privacy directive, towards a third generation of data protection legislations. In *Data Protection in a Profile World*, pages 3–29. Springer, 2010.

[46] A. Rezgui, A. Bouguettaya, and M. Y. Eltoweissy. Privacy on the web: facts, challenges, and solutions. *IEEE Security and Privacy*, pages 40–49, 2003.

[47] A. Rial and G. Danezis. Privacy-preserving smart metering. In *Proceedings of the 2011 ACM Workshop on Privacy in the Electronic Society, WPES 2011.* ACM, 2011.

[48] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[49] M. C. Tschantz, D. K. Kaynar, and A. Datta. Formal verification of differential privacy for interactive systems. *CoRR*, abs/1101.2819, 2011.

[50] M. C. Tschantz and J. M. Wing. Formal methods for privacy. In *FM*, pages 1–15, 2009.

[51] D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. A. Hendler, and G. J. Sussman. Information accountability. *Commun. ACM*, 51(6):82–87, 2008.

[52] T. Yu, N. Li, and A. I. Antón. A formal semantics for P3P. In *SWS*, pages 1–8, 2004.

[53] S. Zdancewic, L. Zheng, N. Nystrom, and A. C. Myers. Secure program partitioning. *ACM Trans. Comput. Syst.*, 20(3):283–328, 2002.

ANNEX 1

In this Annex, we provide some complementary definitions and the proof of Theorem 1 (Section 3). We start with the definition of consistent sequences of events suggested in Subsection 3.2.

Definition 8 A trace $\sigma \in \Theta$ of length n is consistent if and only if:

$$\forall i \in [1,n], \forall j \in [1,n], i \neq j, \sigma_i \in P_A(X) \Rightarrow \sigma_j \notin P_A(X)$$
$$\forall i \in [1,n], \sigma_i \in Open_A^B(X,V) \Rightarrow \exists j \in [1,n], j < i, \sigma_j = Commit_A^B(X,V)$$
$$\forall i \in [1,n], \sigma_i \in Send_A^B(X,V) \Rightarrow \exists j \in [1,n], j < i, \sigma_j = Ev_B(X,V)$$
$$\forall i \in [1,n], \sigma_i \in Commit_A^B(X,V) \Rightarrow \exists j \in [1,n], j < i, \sigma_j = Ev_B(X,V)$$
$$\forall i \in [1,n], \sigma_i \in Get_A(X,V) \Rightarrow \exists j \in [1,n], j < i, \sigma_j = Compute_B(X,V)$$

with $P_A(X) = \{Compute_A(X,V), Get_A(X,V), Send_A^B(X,V), Commit_A^B(X,V) \mid V \in Val\}, B \in \Omega$ and $Ev_B(X,V) = Compute_B(X,V)$ *or* $Ev_B(X,V) = Send_B^C(X,V)$ for any actor C.

As suggested in Subsection 3.2, a consistent sequence of events is a sequence of events in which variables are used consistently: a variable cannot be both computed and received by an actor, is received or computed only once, can be opened only to the value corresponding to an earlier commitment and sent only if it has been previously received or computed. Focusing on consistent sequences is not a restriction of the framework because, as illustrated in Section 4, any tampering with the variables or malicious action from an actor can be expressed through the *Compute* operations which are completely unrestricted: the value V in a $Compute_A(X,V)$ event does not need to satisfy the equation defining X in Σ.

Before embarking on the proof of Theorem 1, we introduce a variant of Definition 1 which defines the global state after a trace σ. The intuition is that the state $S(\sigma, \eta_0)$ includes the values of all variables (in all local states of all actors) after the execution of σ. As for Definition 1, by abuse of notation, we write $S(\sigma)$ for $S(\sigma, \emptyset)$ in the sequel, with $\emptyset$ the empty environment.

$$\begin{aligned} S(\langle\rangle, \eta) &= \eta \\ S(e.\sigma, \eta) &= S(\sigma, T(e, \eta)) \end{aligned}$$

$$\begin{aligned} T(Compute_A(X,V), \eta) &= \eta[V/X] \\ T_A(Ev, \eta) &= \eta \quad for\ Ev \neq Compute_A(X,V) \end{aligned}$$

The intuition is that the only events adding some information to the global state are the *Compute* events. The effect of all other events is the communication of information about a variable from an actor to another one.

An interesting benefit of trace consistency is that a variable cannot take two different values for two different actors. The only difference between the states of two different actors is that some variables may be defined in one state and undefined or defined only in terms of a committed value in the other state. In other words, all local states are consistent approximations of the global state:

Lemma 1 $\forall \sigma \in \Theta, \forall A \in \Omega, S_A(\sigma) \leqslant S(\sigma)$

Definition 9 $\forall \eta \in St, \forall \eta' \in St, \eta \leqslant \eta'$ if and only if $\forall X \in Var$,

$$\begin{aligned} \eta(X) &= \eta'(X) \ \ or \\ \eta(X) &= \Xi(\eta'(X)) \ \ or \\ \eta(X) &= \perp \end{aligned}$$

Lemma 1 can be proved by induction on the length of sequences σ (using Definition 8 to show that no additional event in the sequence can introduce any inconsistency between the local states of the actors). Let us now show further intermediate results which are useful for the proof of Theorem 1.

Lemma 2 $\forall U \in Context, \forall U' \in Context, \forall T \in Vars, \forall X \in Var, U \subseteq U'$ and $U \vdash_T X \Rightarrow U' \vdash_T X$

Lemma 2 can be proved by induction on the derivation tree of $U \vdash_T X$.

Lemma 3 $\forall A \in \Omega, \forall U \in Context, \forall U' \in Context, \forall \sigma \in \Theta, U \subseteq U'$ and $Complete_A(U', \sigma) \Rightarrow Complete_A(U, \sigma)$

Lemma 3 follows directly from Definition 3.

Lemma 4 $\forall A \in \Omega, \forall U \in Context, \forall \sigma \in \Theta, \forall X \in Var, \forall X' \in Var, X' \in Dep(X)$ and $Detect_A^U(\sigma, X') \Rightarrow Detect_A^U(\sigma, X)$

Lemma 5 $\forall A \in \Omega, \forall U \in Context, \forall U' \in Context, U \subseteq U', \forall \sigma \in \Theta, \forall X \in Var, \forall X' \in Var, Detect_A^U(\sigma, X) \Rightarrow Detect_A^{U'}(\sigma, X)$

Lemma 4 and Lemma 5 follow directly from Definition 7. We can now prove Theorem 1 by induction of the tree expression defining variable X in Σ (which amounts to an induction on the derivation tree for statements $U \vdash_T X$) considering each rule of Definition 2 in turn.

Proof of Theorem 1:

Rule R1:

We assume that Theorem 1 holds for each variable Y_i (Induction Hypothesis) and show that it holds for X.

Let $\sigma \in \Theta$ such that $Complete_A(U, \sigma)$ and $Incorrect(\sigma, X)$. From $X \in T$ and the trust assumption (Definition 6), we have $\exists i, Incorrect(\sigma, Y_i)$. By Lemma 3, we also have $Complete_A(U_i, \sigma)$. Applying the Induction Hypothesis, we can thus derive $Detect_A^{U_i}(\sigma, Y_i)$. From Lemma 5 we then have $Detect_A^U(\sigma, Y_i)$ and Lemma 4 allows us to derive $Detect_A^U(\sigma, X)$ which concludes the proof for Rule R1.

Rule R2:

We assume that Theorem 1 holds for each variable Y_i (Induction Hypothesis) and show that it holds for X.

Let $\sigma \in \Theta$ such that $Complete_A(U, \sigma)$ and $Incorrect(\sigma, X)$. Let E_1 and E_2 be the subsets of $\{X, Y_1, \ldots, Y_n\}$ defined as follows:

$E_1 = \{Z \in \{X, Y_1, \ldots, Y_n\} \mid S_A(\sigma)(Z) = \perp\}$

$E_1 = \{Z \in \{X, Y_1, \ldots, Y_n\} \mid \exists V \in Val, S_A(\sigma)(Z) = \Xi(V)\}$

Let $\sigma' = \sigma.Get(Z_1^1) \ldots (Z_{n_1}^1).Open(Z_1^2) \ldots (Z_{n_2}^2)$

with $E_1 = \{Z_1^1 \ldots Z_{n_1}^1\}$ and $E_2 = \{Z_1^2 \ldots Z_{n_2}^2\}$

Let $E = S_A(\sigma)$ and $E' = S_A(\sigma')$

Case 1: $E'(X) \neq F(E'(Y_1), \ldots, E'(Y_n))$

From $Complete_A(U, \sigma)$, we can infer:

$\forall Z \in E_1, \forall (R, O, G) \in U, Z \notin R$ and $Z \notin O$, hence $Z \in G$ and $\forall Z \in E_2, \forall (R, O, G) \in U, Z \notin R$ and $Z \notin G$, hence $Z \in O$, which allows us to prove $Detect_A^U(\sigma, X)$[8].

Case 2: $E'(X) = F(E'(Y_1), \ldots, E'(Y_n))$

We also have $E'(X) = S_A(\sigma')(X) = S(\sigma')(X) = S(\sigma)(X)$ from Lemma 1, the definition of $\leqslant$ and S respectively. Similarly $E'(Y_i) = S_A(\sigma')(Y_i) = S(\sigma')(Y_i) = S(\sigma)(Y_i)$. From $Incorrect(\sigma, X)$, we have $S(\sigma)(X) \neq Eval(X, S(\sigma))$, which allows us to derive $\exists i, S(\sigma)(Y_i) \neq Eval(Y_i, S(\sigma))$ and, therefore $\exists i, Incorrect(\sigma, Y_i)$. By Lemma 3, we also have $Complete_A(U_i, \sigma)$.

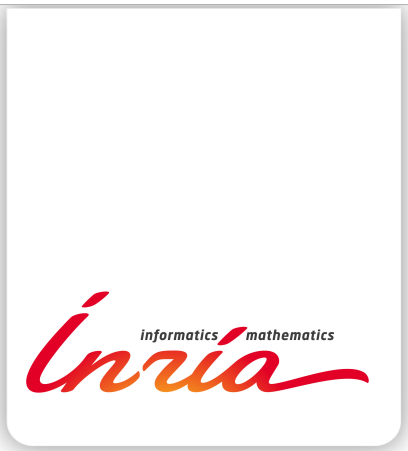
[8]With Z in the definiton of *Detect* instantiated to X.

Applying the Induction Hypothesis, we can thus derive $Detect_A^{U_i}(\sigma, Y_i)$. From Lemma 5 we then have $Detect_A^U(\sigma, Y_i)$ and Lemma 4 allows us to conclude $Detect_A^U(\sigma, X)$ which concludes the proof for Rule R2.

Rule R3:

From the definition of S and $Eval$, we cannot have $Incorrect(\sigma, X)$ with X an input variable ($X \in In$) because $Eval(X, S(\sigma)) = S(\sigma)(X)$, which concludes the proof for Rule R3 and for Theorem 1.

Contents



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399