Privacy Architectures: Reasoning About Data Minimisation and Integrity

Thibaud Antignac, Daniel Le Métayer

Privacy Architectures: Reasoning About Data Minimisation and Integrity*

Thibaud Antignac and Daniel Le Métayer

Inria, University of Lyon, France
{thibaud.antignac,daniel.le-metayer}@inria.fr

Abstract. Privacy by design will become a legal obligation in the European Community if the Data Protection Regulation eventually gets adopted. However, taking into account privacy requirements in the design of a system is a challenging task. We propose an approach based on the specification of privacy architectures and focus on a key aspect of privacy, data minimisation, and its tension with integrity requirements. We illustrate our formal framework through a smart metering case study.

1 Introduction

The philosophy of privacy by design is that privacy should not be treated as an afterthought but as a first-class requirement in the design of IT systems. Privacy by design will become a legal obligation in the European Community if the Data Protection Regulation [11] eventually gets adopted. However, from a technical standpoint privacy by design is a challenging endeavour: first, privacy is a multi-faceted notion stemming from a variety of principles[1] which are generally not defined very precisely; in addition, these requirements may be (or may seem to be) in tension with other requirements such as functional requirements, ease of use or performances. To implement these requirements, a wide array of privacy enhancing technologies (PETs) are available[2]. Each of these techniques provides different guarantees based on different assumptions and therefore is suitable in different contexts. As a result, it is quite complex for a software engineer to make informed choices among all these possibilities and to find the most appropriate combination of techniques to solve his own requirements. Solutions have been proposed in different application domains such as smart metering [14,29], pay-as-you-drive [2,18], or location-based systems [20] but the next challenge in this area is to go beyond individual cases and to establish sound foundations and methodologies for privacy by design [9,33]. In this paper, we advocate the idea that privacy by design should be addressed at the architectural level, because it

* The final publication is available at link.springer.com (URL not yet available).

[1] These principles include collection limitation, data quality, purpose specification, use limitation, security, openness, individual participation, accountability, etc.

[2] For example homomorphic encryption, zero-knowledge proof, secure multi-party computation, private information retrieval, anonymous credentials, anonymous communication channels, etc.

makes it possible to abstract away unnecessary details, and should be supported by a formal model. The fact that not all aspects of privacy are susceptible to formalisation is not a daunting obstacle to the use of formal methods for privacy by design: the key issue is to be able to build appropriate models for the aspects of privacy that are prone to formalisation and involve complex reasoning. Data minimisation, which is one of the key principles of most privacy guidelines and regulations, is precisely one of these aspects. Data minimisation stipulates that the collection and processing of personal data should always be done with respect to a particular purpose and the amount of data strictly limited to what is really necessary to achieve the purpose [11].

In this paper, data minimisation requirements are expressed as properties defining for each stakeholder the information that he is (or is not) allowed to know. Data minimisation would not be so difficult to achieve if other, sometimes conflicting, requirements did not have to be met simultaneously. Another common requirement, which we call "integrity" in the sequel, is the fact that some stakeholders may require guarantees about the correctness of the result of a computation. In fact, the tension between data minimisation and integrity is one of the delicate issues to be solved in many systems involving personal data.

In Section 2 we propose a language to define privacy architectures. In Section 3, we introduce a logic for reasoning about architectures and show the correctness and completeness of its axiomatisation. This axiomatisation is used in Section 4 to prove that an example of smart metering architecture meets the expected privacy and minimisation requirements. Section 5 discusses related work and Section 6 outlines directions for further research.

2 Privacy Architectures

Many definitions of architectures have been proposed in the literature. In this paper, we adopt a definition inspired by [4][3]: *The architecture of a system is the set of structures needed to reason about the system, which comprise software and hardware elements, relations among them and properties of both.* The atomic components of an architecture are coarse-grain entities such as modules, components or connectors. In the context of privacy, the components are typically the PETs themselves and the purpose of the architecture is their combination to achieve the requirements of the system.

The meaning of the requirements considered here (minimisation and integrity) depends on the purpose of the data collection, which is equated to the expected functionality of the system here. In the sequel, we assume that this functionality is expressed as the computation of a set of equations[4] Ω such that $\Omega = \left\{\tilde{X} = T\right\}$ with terms T defined as shown in Table 1. $\tilde{X}$ represents (potentially indexed) variables and X simple variables ($X \in \mathit{Var}$), k index variables

[3] This definition is a generalisation (to system architectures) of the definition of software architectures proposed in [4].

[4] Which is typically the case for systems involving integrity requirements.

($k \in \mathit{Index}$), Cx constants ($Cx \in \mathit{Const}$), Ck index constants ($Ck \in \mathbb{N}^5$), F functions ($F \in \mathit{Fun}$) and $\odot F(X)$ is the iterative application of function F to the elements of the array denoted by X (e.g. sum of the elements of X if F is equal to $+$). We assume that each array variable X represents an array of fixed size $\mathit{Range}(X)$.

$$
\begin{aligned}
T &::= \tilde{X} \mid Cx \mid F(T_1, \ldots, T_n) \mid \odot F(X) \\
\tilde{X} &::= X \mid X_K \\
K &::= k \mid Ck
\end{aligned}
$$

Table 1. Term Language.

In the following subsections, we introduce our privacy architecture language (Subsection 2.1) and its semantics (Subsection 2.2).

2.1 Privacy Architecture Language

We define an architecture as a set of components C_i, $i \in [1, \ldots, n]$ associated with relations describing their capabilities. These capabilities depend on the set of available PETs. For the purpose of this paper, we consider the architecture language described in Table 2.

$$
\begin{aligned}
A &::= \{R\} \\
R &::= \mathit{Has}_i\left(\tilde{X}\right) && \mid \mathit{Receive}_{i,j}\left(\{S\}, \{\tilde{X}\}\right) \\
&\mid \mathit{Compute}_i\left(\tilde{X} = T\right) && \mid \mathit{Check}_i\left(\{Eq\}\right) \\
&\mid \mathit{Verif}_i^{\mathit{Proof}}\left(\mathit{Pro}\right) && \mid \mathit{Verif}_i^{\mathit{Attest}}\left(\mathit{Att}\right) \\
&\mid \mathit{Spotcheck}_{i,j}\left(X_k, Eq\right) && \mid \mathit{Trust}_{i,j} \\
\\
S &::= \mathit{Pro} \mid \mathit{Att} && \mathit{Att} ::= \mathit{Attest}_i\left(\{Eq\}\right) \\
\mathit{Pro} &::= \mathit{Proof}_i\left(\{P\}\right) && \mathit{Eq} ::= T_1\ \mathit{Rel}\ T_2 \\
P &::= \mathit{Att} \mid \mathit{Eq} && \mathit{Rel} ::= = \mid < \mid > \mid \leq \mid \geq
\end{aligned}
$$

Table 2. Privacy Architecture Language.

Subscripts i and j are component indexes and the notation $\{Z\}$ is used to define a set of terms of category Z. $\mathit{Has}_i(\tilde{X})$ expresses the fact that variable $\tilde{X}$ is an input variable located at component C_i (e.g. sensor or meter) and $\mathit{Receive}_{i,j}(\{S\}, \{\tilde{X}\})$ specifies that component C_i can receive from component C_j messages consisting of a set of statements $\{S\}$ and a set of variables $\{\tilde{X}\}$. A statement can be either a proof of a set of properties P (denoted by $\mathit{Proof}_i\left(\{P\}\right)$) or an attestation (denoted by $\mathit{Attest}_i(\{Eq\})$), that is to say a simple declaration

[5] Set of natural numbers.

by a component C_i that properties Eq are true. A component can also compute a variable defined by an equation $\tilde{X} = T$ (denoted by $Compute_i(\tilde{X} = T)$), check that a set of properties Eq holds (denoted by $Check_i(\{Eq\})$), verify a proof of a property Pro received from another component (denoted by $Verif_i^{Proof}(Pro)$), verify the origin of an attestation (denoted by $Verif_i^{Attest}(Att)$), or perform a spotcheck. A spotcheck, which is denoted by $Spotcheck_{i,j}(X_k, Eq)$, is the request from a component C_j of a value X_k taken from array X and the verification that this value satisfies property Eq. Primitive properties Eq are simple equations on terms T. Last but not least, trust assumptions are expressed using $Trust_{i,j}$ (meaning that component C_i trusts component C_j). In the sequel, we use Γ to denote the set of architectures following the syntax of Table 2. Architectures can also be defined using graphical representations. As an illustration, Figure 1 displays a simple architecture involving a meter M and the central server of a provider P. The meter plays both the role of a sensor providing the input consumption values ($Has_M(Cons_t)$) and the role of a secure element computing the fee. Because the provider trusts the meter ($Trust_{P,M}$), it merely checks the certificate $Attest_M(\{Fee = \odot + (y), y_t = F(x_t), x_t = S(Cons_t)\})$ sent by the meter.

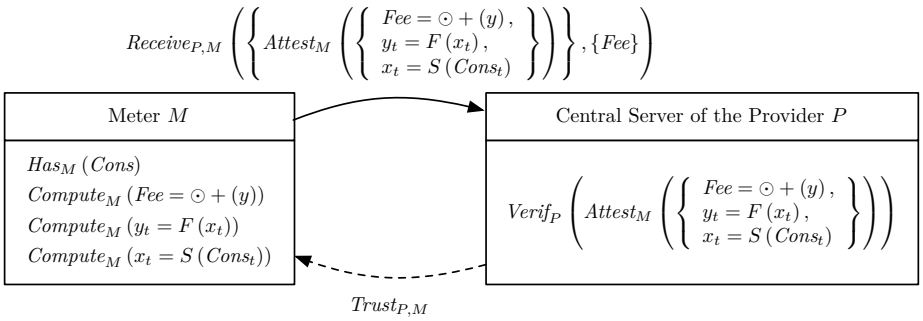

Fig. 1. Example of smart metering architecture.

Strictly speaking, we should introduce a notion of actor and a relationship between actors and the components that are under their control but, for the sake of brevity (and without loss of generality[6]), we do not distinguish between components and actors here.

Architectures provide an abstract, high-level view of a system: for example, we do not express at this level the particular method (for example, a zero-knowledge proof protocol) used by a component to build a proof ($Proof_i(\{P\})$) or to verify it, or to check that another component has actually certified (attested) a property ($Verif_i^{Attest}(Att)$). Another main departure from protocol specification languages is that we do not have any specific ordering or notion of sequentiality here, even though functional dependencies introduce implicit constraints in the

[6] The fact that an actor controls several components can be expressed through a trust relationship.

events of the system, as discussed below. The objective is to express and reason about the main design choices rather than to cover all the development steps.

2.2 Privacy Architectures Semantics

The definition of the semantics of an architecture is based on its set of compatible traces. A trace is a sequence of high-level events occurring in the system as presented in Table 3. Events can be seen as instantiated relations of the architecture. For example, a $Receive_{i,j}\left(\{S\},\{\tilde{X}:V\}\right)$ event specifies the values V of the variables $\tilde{X}$ received by C_i. Similarly, $Spotcheck_{i,j}\left(X_{Ck}:V,\{Eq\}\right)$ specifies the specific index Ck (member of $\mathbb{N}$) chosen by C_i for the spotcheck and the value V of X_{Ck}. All variable indexes occurring in events, except for variables occurring in the properties of $Receive_{i,j}$, $Verif_i^{Proof}$, $Verif_i^{Attest}$, and $Spotcheck_{i,j}$, must belong to $\mathbb{N}$.

$$
\begin{array}{lll}
\theta ::= Seq(\epsilon) & & \\
\epsilon ::= Has_i\left(\tilde{X}:V\right) & \mid Receive_{i,j}\left(\{S\},\{\tilde{X}:V\}\right) \\
\quad \mid Compute_i\left(\tilde{X}=T\right) & \mid Check_i\left(\{Eq\}\right) \\
\quad \mid Verif_i^{Proof}\left(Pro\right) & \mid Verif_i^{Attest}\left(Att\right) \\
\quad \mid Spotcheck_{i,j}\left(X_{Ck}:V,\{Eq\}\right) &
\end{array}
$$

Table 3. Events and traces.

In the following, we consider only consistent architectures and consistent traces. An architecture is said to be consistent if each variable can be computed (or can be initially possessed, as expressed by Has_i) by a single component, a component cannot receive a variable from different sources, a component computing a variable or checking a property can receive or compute all the necessary input variables (variables occuring in T for $Compute_i(\tilde{X}=T)$, in Eq for $Check_i(Eq)$), a component can only verify properties that it can receive from another component, etc. The same kind of consistency assumptions apply to traces, in addition to ordering consistency properties (variables and properties are not used before being received or computed).

We use *Event* to denote the set of events ϵ and *Trace* to denote the set of consistent traces θ.

Definition 1 (Compatibility). *A trace θ of length $\overline{\theta}$ is compatible with an architecture A if and only if:*

$$
\begin{array}{l}
\forall a\in[1,\overline{\theta}],\ \textit{if } \theta_a \neq Compute_i\left(\tilde{X}=T\right)\ \textit{ then } \exists \alpha\in A, \mathcal{C}(\theta_a,\alpha)\ \textit{ and} \\
\qquad \textit{if } \theta_a = Spotcheck_{i,j}\left(X_{Ck}:V,\{Eq\}\right) \\
\qquad\qquad \textit{then } \forall b\in[1,\overline{\theta}], b\neq a \Rightarrow \forall k',V',Eq', \\
\qquad\qquad\qquad\qquad \theta_b \neq Spotcheck_{i,j}\left(X_{k'}:V',\{Eq'\}\right)
\end{array}
$$

where $\mathcal{C}(\epsilon, \alpha)$ holds if and only if ϵ can be obtained from α by adding specific values V for variables and instantiating index variables to integer values.

The first condition in the definition of compatibility states that, except for compute events, only events which are instantiations of components of the architecture A can appear in the trace θ. The rationale for excepting compute events is the need to express the potential actions of a curious agent trying to derive the value of a variable $\tilde{X}$ from the values of variables that he already has. As a result, compatible traces may include computations that are not contemplated by the architecture, provided that the component possesses all the variables necessary to perform this computation (consistency assumption). The adversary model considered here includes computation of new variables, erroneous computations, and communication of incorrect values, which corresponds to Dolev-Yao attacks for internal stakeholders (except they cannot break the protocol). The second condition expresses the fact that spotchecks can be performed only once. This condition could be relaxed through the introduction of an additional threshold parameter t to express the fact that up to t spotchecks are possible. We denote by $T(A)$ the set of compatible traces of an architecture A.

In order to define the semantics of events, we introduce first the notion of state of a component:

$$\begin{aligned} State &= (State_V \times State_P \times State_P) \cup \{Error\} \\ State_V &= (Var \rightarrow Val_\perp) \\ State_P &= \{\{Eq\} \cup \{Trust_{i,j}\}\} \end{aligned}$$

The state of a component is either the error state *Error* or a triple made of a variable state assigning a value (or the undefined value $\perp$[7]) to each variable and two property states: the first one defines the set of properties known by the component and the second one the set of properties believed by the component (after a spotcheck). In the sequel, we use σ to denote the global state (state of the components $\langle C_1, \ldots, C_n \rangle$) defined on $State^n$. The initial state for an architecture A is denoted by $Init^A = \langle Init_1^A, \ldots, Init_n^A \rangle$ with:

$$\forall i \in [1, n], Init_i^A = (Empty, \{Trust_{i,j} | Trust_{i,j} \in A\}, \emptyset)$$

where *Empty* denotes the empty variable state ($\forall X \in Var, Empty(X) = \perp$). The only information contained in the initial state is the trust properties specified by the architecture.

The semantics function S_T is defined in Table 4. It specifies the impact of a trace on the state of each component C_i. It is defined as an iteration through the trace with function S_E defining the impact of each type of event on the states of the components.

The notation $\epsilon.\theta$ is used to denote a trace whose first element is ϵ and the rest of the trace is θ. Each event modifies only the state of the component C_i.

[7] Please note that $\perp$ is used to denote undefined values, that is to say values which have not been set, as opposed to error values (e.g. division by zero or type errors). We do not consider computation error values here.

$$
\begin{aligned}
S_T :\ & \mathit{Trace} \times \mathit{State}^n \to \mathit{State}^n \\
S_E :\ & \mathit{Event} \times \mathit{State}^n \to \mathit{State}^n \\
S_T\left(\langle\rangle, \sigma\right) &= \sigma \\
S_T\left(\epsilon.\theta, \sigma\right) &= S_T(\theta, S_E(\epsilon, \sigma)) \\
S_E\left(\mathit{Has}_i\left(\tilde{X} : V\right), \sigma\right) &= \sigma[\sigma_i/(\sigma_i^v[\tilde{X}/V], \sigma_i^{pk}, \sigma_i^{pb})] \\
S_E\left(\mathit{Receive}_{i,j}\left(\{S\}, \{\tilde{X} : V\}\right), \sigma\right) &= \sigma[\sigma_i/(\sigma_i^v[\{\tilde{X}/V\}], \sigma_i^{pk}, \sigma_i^{pb})] \\
S_E\left(\mathit{Compute}_i\left(\tilde{X} = T\right), \sigma\right) &= \sigma[\sigma_i/(\sigma_i^v[\tilde{X}/\varepsilon(T, \sigma_i^v)], \sigma_i^{pk} \cup \{\tilde{X} = T\}, \sigma_i^{pb})] \\
S_E\left(\mathit{Check}_i\left(E\right), \sigma\right) &= \sigma[\sigma_i/(\sigma_i^v, \sigma_i^{pk} \cup E, \sigma_i^{pb})] \\
&\qquad \text{if } \forall Eq \in E, \varepsilon(Eq, \sigma_i^v) = \mathit{True} \\
&= \sigma[\sigma_i/\mathit{Error}] \text{ otherwise} \\
S_E\left(\mathit{Verif}_i^{\mathit{Proof}}\left(\mathit{Proof}_j(E)\right), \sigma\right) &= \sigma[\sigma_i/(\sigma_i^v, \sigma_i^{pk} \cup \{Eq | Eq \in E \text{ or} \\
&\qquad\qquad (\mathit{Attest}_{j'}(E') \in E \text{ and} \\
&\qquad\qquad Eq \in E' \text{ and} \\
&\qquad\qquad \mathit{Trust}_{i,j'} \in \sigma_i^{pk})\}, \sigma_i^{pb})] \\
&\qquad \text{if } \overline{\mathit{Verif}}^{\mathit{Proof}}((E), \sigma_i^v) = \mathit{True} \\
&= \sigma[\sigma_i/\mathit{Error}] \text{ otherwise} \\
S_E\left(\mathit{Verif}_i^{\mathit{Attest}}\left(\mathit{Attest}_j(E)\right), \sigma\right) &= \sigma[\sigma_i/(\sigma_i^v, \sigma_i^{pk} \cup \{Eq | Eq \in E \text{ and} \\
&\qquad\qquad \mathit{Trust}_{i,j} \in \sigma_i^{pk}\}, \sigma_i^{pb})] \\
&\qquad \text{if } \overline{\mathit{Verif}}^{\mathit{Attest}}((E), \sigma_i^v) = \mathit{True} \\
&= \sigma[\sigma_i/\mathit{Error}] \text{ otherwise} \\
S_E\left(\mathit{Spotcheck}_{i,j}\left(X_{Ck} : V, E\right), \sigma\right) &= \sigma[\sigma_i/(\sigma_i^v[X_{Ck}/V], \sigma_i^{pk}, \sigma_i^{pb} \cup E)] \\
&\qquad \text{if } \forall Eq \in E, \\
&\qquad\quad \varepsilon(Eq[k/Ck], \sigma_i^v[X_{Ck}/V]) = \mathit{True} \\
&= \sigma[\sigma_i/\mathit{Error}] \text{ otherwise}
\end{aligned}
$$

Table 4. Semantics of traces of events.

This modification is expressed as $\sigma[\sigma_i/(v, pk, pb)]$ (or $\sigma[\sigma_i/\mathit{Error}]$ in the case of the error state) that replaces the variable and property components of the state of C_i by v, pk, and pb respectively. We assume that no event $\theta_{a'}$ with $a' > a$ involves component C_i if its state σ_i is equal to *Error* after the occurrence of θ_a (in other words, any error in the execution of a component causes this component to stop).

The effect of Has_i and $\mathit{Receive}_{i,j}$ on the variable state of component C_i is the replacement of the values of the variables $\tilde{X}$ by new values $V \in \mathit{Val}$, which is denoted by $\sigma_i^v[\tilde{X}/V]$.

The effect of $Compute_i(\tilde{X} = T)$ is to set the variable $\tilde{X}$ to the evaluation of the value of T in the current variable state σ_i^v of C_i, which is defined by $\varepsilon(T, \sigma_i^v)$. $Spotcheck_i(X_{Ck} : V, E)$ sets the value of X_{Ck} to V. The other events do not have any effect on the variable state of C_i. The value of a variable replaced after the occurrence of an event must be $\perp$ before its occurrence[8]. We assume that it is different from $\perp$ and does not involve any $\perp$) after the event[9].

Most events also have an effect on the property states. This effect is the addition to the property states of the new knowledge or belief provided by the event. For $Compute_i\left(\tilde{X} = T\right)$, this new knowledge is the equality $\tilde{X} = T$; for the $Check_i$, $Verif_i$, and $Spotcheck_{i,j}$ events, the new knowledge is the properties checked or verified. In all cases except for $Spotcheck_{i,j}$ these properties are added to the pk property state because they are known to be true by component C_i; in the case of $Spotcheck_{i,j}$ the properties are added to the pb property state because they are believed by C_i: they have been checked on a sample value X_{Ck} but might still be false for some other X_k. The only guarantee provided to C_i by $Spotcheck_{i,j}$ is that C_i has always the possibility to detect an error (but he has to choose an appropriate index, that is to say an index that will reveal the error).

Functions $\overline{Verif}^{Proof}$ and $\overline{Verif}^{Attest}$ define the semantics of the corresponding verification operations. As discussed above, we do not enter into the internals of the proof and attestation verifications here and just assume that only true properties are accepted by $\overline{Verif}^{Proof}$ and only attestations provided by the authentic sender are accepted by $\overline{Verif}^{Attest}$. The distinctive feature of $Verif_i^{Attest}$ events is that they generate new knowledge only if the author of the attestation can be trusted (hence the $Trust_{i,j} \in A$ condition).

Let us note also that $Receive_{i,j}$ events do not add any new knowledge by themselves because the received properties have to be verified before they can be added to the property states.

We can now define the semantics of an architecture A as the set of the possible states produced by compatible traces.

Definition 2 (Semantics of architectures.). *The semantics of an architecture A is defined as:* $\mathcal{S}(A) = \{\sigma \in State^n \mid \exists \theta \in T(A), S_T(\theta, Init^A) = \sigma\}$.

In the following, we use $\mathcal{S}_i(A)$ to denote the subset of $\mathcal{S}(A)$ containing only states which are well defined for component C_i: $\mathcal{S}_i(A) = \{\sigma \in \mathcal{S}(A) \mid \sigma_i \neq Error\}$. The prefix ordering on traces gives rise to the following ordering on states: $\forall \sigma \in \mathcal{S}_i(A), \forall \sigma' \in \mathcal{S}_i(A), \sigma \geq_i \sigma' \Leftrightarrow \exists \theta \in T(A), \exists \theta' \in T(A), \sigma = S_T(\theta, Init^A), \sigma' = S_T(\theta', Init^A)$, and θ' is a prefix of θ.

[8] Because we consider only consistent traces. A value different from $\perp$ would mean that the variable is computed or set more than once.

[9] In other words, input values and results of computations are fully defined.

3 Privacy Logic

Because privacy is closely connected with the notion of knowledge, epistemic logics form an ideal basis to reason about privacy properties. Epistemic logics [12] are a family of modal logics using a knowledge modality usually denoted by $K_i(\psi)$ to denote the fact that agent i knows the property ψ. However standard epistemic logics based on possible worlds semantics suffer from a weakness which makes them unsuitable in the context of privacy: this problem is often referred to as "logical omniscience" [17]. It stems from the fact that agents know all the logical consequences of their knowledge (because these consequences hold in all possible worlds). An undesirable outcome of logical omniscience would be that, for example, an agent knowing the hash $H(v)$ of a value v would also know v. This is obviously not the intent in a formal model of privacy where hashes are precisely used to hide the original values to the recipients. This issue is related to the fact that standard epistemic logics do not account for limitations of computational power.

Therefore it is necessary to define dedicated epistemic logics to deal with different aspects of privacy and to model the variety of notions at hand (e.g. knowledge, zero-knowledge proof, trust, etc.). In this paper, we follow the "deductive algorithmic knowledge" approach [12,28] in which the explicit knowledge of a component C_i is defined as the knowledge that this component can actually compute using his own deductive system $\triangleright_i$. The deductive relation $\triangleright_i$ is defined here as a relation between a set of Eq and Eq properties: $\{Eq_1, \ldots, Eq_n\} \triangleright_i Eq_0$. Typically, $\triangleright_i$ can be used to capture properties of the functions of the specification. For example $\{h_1 = H(x_1), h_2 = H(x_2), h_1 = h_2\} \triangleright_i (x_1 = x_2)$ expresses the injectivity property of a hash function H. Another relation, Dep_i, is introduced to express that a variable can be derived from other variables. $Dep_i\left(\tilde{X}, \left\{\tilde{X}^1, \ldots \tilde{X}^n\right\}\right)$ means that a value for $\tilde{X}$ can be obtained by C_i $(\exists F, \tilde{X} = F(\tilde{X}^1, \ldots, \tilde{X}^n))$. The absence of a relation such as $Dep_i(x_k, \{y_k\})$ prevents component C_i from deriving the value of x_k from the value of y_k, capturing the hiding property of the hash application $y_k = H(x_k)$.

$$
\begin{aligned}
\phi &::= Has_i^{all}\left(\tilde{X}\right) \mid Has_i^{none}\left(\tilde{X}\right) \mid Has_i^{one}\left(\tilde{X}\right) \\
&\quad \mid K_i(Eq) \quad \mid B_i(Eq) \quad \mid \phi_1 \wedge \phi_2 \\
Eq &::= T_1\ Rel\ T_2 \mid Eq_1 \wedge Eq_2
\end{aligned}
$$

Table 5. Architecture logic.

This logic involves two modalities, denoted by K_i and B_i, which represent respectively knowledge and belief properties of a component C_i. Please note that the Eq notation (already used in the language of architectures) is overloaded, without ambiguity: it is used to denote conjunctions (rather than sets) of primitive relations in the logic. The logic can be used to express useful properties of architectures: for example $Has_i^{all}(\tilde{X})$ expresses the fact that component C_i can obtain or derive (using its deductive system $\triangleright_i$) the value of $\tilde{X}_k$ for all k

in $Range(X)$. $Has_i^{one}(\tilde{X})$ expresses the fact that component C_i can obtain or derive the value of X_k for at most one k in $Range(X)$. Finally, $Has_i^{none}(\tilde{X})$ is the privacy property stating that C_i does not know any X_k value. It should be noted that Has_i properties only inform on the fact that C_i can get or derive some values for the variables but they do not bring any guarantee about the correctness of these values. Such guarantees can only be ensured through integrity requirements, expressed using the $K_i(Eq)$ and $B_i(Eq)$ properties. $K_i(Eq)$ means that component C_i can establish the truthfulness of Eq while $B_i(Eq)$ expresses the fact that C_i may test this truthfulness, and therefore detect its falsehood or believe that the property is true otherwise.

We can now define the semantics of a property ϕ.

Definition 3 (Semantics of properties). *The semantics $S(\phi)$ of a property ϕ is defined in Table 6 as the set of architectures meeting ϕ.*

$$
\begin{aligned}
A \in S\left(Has_i^{all}\left(\tilde{X}\right)\right) &\Leftrightarrow \exists \sigma \in \mathcal{S}(A), \sigma_i^v(\tilde{X}) \text{ does not contain any } \bot \\
A \in S\left(Has_i^{none}\left(\tilde{X}\right)\right) &\Leftrightarrow \forall \sigma \in \mathcal{S}(A), \sigma_i^v(\tilde{X}) = \bot \\
A \in S\left(Has_i^{one}\left(\tilde{X}\right)\right) &\Leftrightarrow \forall \sigma \in \mathcal{S}(A), \sigma_i^v(\tilde{X}) = \bot \vee (\sigma_i^v(\tilde{X}) =< v_1, \ldots, v_k > \wedge \\
&\qquad \nexists (u, u'), u \neq u' \wedge v_u \neq \bot \wedge \\
&\qquad v_{u'} \neq \bot) \\
A \in S\left(K_i\left(Eq\right)\right) &\Leftrightarrow \forall \sigma' \in \mathcal{S}_i(A), \exists \sigma \in \mathcal{S}_i(A), \exists Eq', (\sigma \geq_i \sigma') \wedge (\sigma_i^{pk} \triangleright_i Eq') \wedge \\
&\qquad (Eq' \Rightarrow Eq) \\
A \in S\left(B_i\left(Eq\right)\right) &\Leftrightarrow \forall \sigma' \in \mathcal{S}_i(A), \exists \sigma \in \mathcal{S}_i(A), \exists Eq'_1, \exists Eq'_2, (\sigma \geq_i \sigma') \wedge \\
&\qquad (\sigma_i^{pb} \triangleright_i Eq'_1) \wedge (\sigma_i^{pk} \triangleright_i Eq'_2) \wedge \\
&\qquad \left(\left(Eq'_1 \wedge Eq'_2\right) \Rightarrow Eq\right) \\
A \in S\left(\phi_1 \wedge \phi_2\right) &\Leftrightarrow A \in S(\phi_1) \wedge A \in S(\phi_2)
\end{aligned}
$$

Table 6. Semantics of properties.

An architecture satisfies the $Has_i^{all}(\tilde{X})$ property if and only if C_i may obtain the full value of $\tilde{X}$ in at least one compatible execution trace whereas $Has_i^{none}(\tilde{X})$ holds if and only if no execution trace can lead to a state in which C_i gets a value of $\tilde{X}$ (or of any part of its content if $\tilde{X}$ is an array variable). $Has_i^{one}(\tilde{X})$ is true if and only if no execution trace can lead to a state in which C_i knows more than one of the values of the array $\tilde{X}$. The validity of $K_i(Eq)$ and $B_i(Eq)$ properties is defined with respect to correct execution traces (with respect to C_i) since an incorrect trace leads to a state in which an error has been detected by the component[10]. The condition $\sigma \geq \sigma'$ is used to discard states corresponding to incomplete traces in which the property Eq has not yet been established. As discussed above, the capacity for a component C_i to derive new knowledge or beliefs is defined by its deductive system $\triangleright_i$.

[10] This is a usual implicit assumption in protocol verification.

In order to reason about architectures and the knowledge of the components, we introduce in Table 7 an axiomatisation of the logic presented in the previous section. The fact that an architecture A satisfies a property ϕ is denoted by $A \vdash \phi$. Axioms (H1-8) and (HNO) are related to properties Has_i while axioms (K1-5) and (K$\wedge$) are related to the knowledge of the components. Axioms (B), (KB), and (B$\wedge$) handle the belief case. Finally, the remaining axioms are structural axioms dealing with the conjunctive operator.

$$\textbf{H1}\frac{Has_i\left(\tilde{X}\right) \in A}{A \vdash Has_i^{all}\left(\tilde{X}\right)} \qquad \textbf{H2}\frac{Receive_{i,j}\left(S, E\right) \in A \quad \tilde{X} \in \{E\}}{A \vdash Has_i^{all}\left(\tilde{X}\right)}$$

$$\textbf{H3}\frac{Compute_i\left(\tilde{X} = T\right) \in A}{A \vdash Has_i^{all}\left(\tilde{X}\right)} \qquad \textbf{H4}\frac{Spotcheck_{i,j}\left(X_k, E\right) \in A}{A \vdash Has_i^{one}\left(X\right)}$$

$$\textbf{H5}\frac{Dep_i\left(\tilde{X}, \left\{\tilde{X}^1, \ldots \tilde{X}^n\right\}\right) \quad \text{for all } l \in [1,n], A \vdash Has_i^{all}\left(\tilde{X}^l\right)}{A \vdash Has_i^{all}\left(\tilde{X}\right)}$$

$$\textbf{H6}\frac{\text{None of the pre-conditions of H1, H2, H3, H4, or H5 holds for } X \text{ or any } X_k}{A \vdash Has_i^{none}\left(\tilde{X}\right)}$$

$$\textbf{H7}\frac{A \vdash Has_i^{all}\left(\tilde{X}\right)}{A \vdash Has_i^{all}\left(X_k\right)} \text{ for all } k \in Range(X) \qquad \textbf{HNO}\frac{A \vdash Has_i^{none}\left(\tilde{X}\right)}{A \vdash Has_i^{one}\left(\tilde{X}\right)}$$

$$\textbf{H8}\frac{A \vdash Has_i^{none}\left(\tilde{X}\right)}{A \vdash Has_i^{none}\left(X_k\right)} \text{ for all } k \in Range(X) \qquad \textbf{K1}\frac{Compute_i\left(\tilde{X} = T\right) \in A}{A \vdash K_i(\tilde{X} = T)}$$

$$\textbf{K3}\frac{Verif_i^{Proof}\left(Proof_j(E)\right) \in A \quad Eq \in E}{A \vdash K_i(Eq)} \qquad \textbf{K2}\frac{Check_i\left(E\right) \in A \quad Eq \in E}{A \vdash K_i(Eq)}$$

$$\textbf{K4}\frac{Verif_i^{Proof}\left(Proof_j(E)\right) \in A \quad Attest_k(E') \in E \quad Trust_{i,k} \in A \quad Eq \in E'}{A \vdash K_i(Eq)}$$

$$\textbf{K5}\frac{Verif_i^{Attest}\left(Attest_j(E)\right) \in A \quad Trust_{i,j} \in A \quad Eq \in E}{A \vdash K_i(Eq)} \qquad \textbf{KB}\frac{A \vdash K_i(Eq)}{A \vdash B_i(Eq)}$$

$$\textbf{K}\wedge\ \frac{A \vdash K_i(Eq_1) \quad A \vdash K_i(Eq_2)}{A \vdash K_i(Eq_1 \wedge Eq_2)} \qquad \textbf{B}\frac{Spotcheck_{i,j}\left(X_k, E\right) \in A \quad Eq \in E}{A \vdash B_i(Eq)}$$

$$\textbf{K}\triangleright\ \frac{E \triangleright_i Eq_0 \quad \text{for all } Eq \in E, A \vdash K_i(Eq)}{A \vdash K_i(Eq_0)} \qquad \textbf{I}\wedge\ \frac{A \vdash \phi_1 \quad A \vdash \phi_2}{A \vdash \phi_1 \wedge \phi_2}$$

$$\textbf{B}\wedge\ \frac{A \vdash B_i(Eq_1) \quad A \vdash B_i(Eq_2)}{A \vdash B_i(Eq_1 \wedge Eq_2)} \qquad \textbf{B}\triangleright\ \frac{E \triangleright_i Eq_0 \quad \text{for all } Eq \in E, A \vdash B_i(Eq)}{A \vdash B_i(Eq_0)}$$

Table 7. Axiomatics.

The axiomatics meets the following soundness, completeness, and decidability properties.

Property 1 (Soundness). *For all A in Γ, if $A \vdash \phi$ then $A \in S(\phi)$.*

The soundness property can be proved by considering each rule in Table 7 in turn and showing that the traces specified in Table 6 have the expected properties (or that appropriate traces can be found in the case of Has_i^{all}).

Property 2 (Completeness). *For all A in Γ, if $A \in S(\phi)$ then $A \vdash \phi$.*

Completeness can be proved by systematic inspection of the different cases in Table 4 that can make a property ϕ true in the trace semantics.

Property 3 (Decidability). *If the deductive systems $\triangleright_i$ are decidable, then the axiomatics is decidable.*

The intuition is that proofs can be stratified into proofs of Has_i^{all}, Has_i^{none}, Has_i^{one}, K_i, and B_i successively, with proofs of properties not involving the deductive systems of the components first and those involving the deductive systems of the components as the last step.

4 Smart Meter Case Study

One of the services provided by smart metering systems is the periodic billing of an amount *Fee* based on the customers consumption $Cons_t$ for periods of time t. The service $Fee = \sum_t (F(S(Cons_t)))$ (where F and S stand for pricing and metering) is expressed as $\Omega = \{Fee = \odot + (y), y_t = F(x_t), x_t = S(Cons_t)\}$. We provide the details for the provider only here but a similar approach could be used for customers or other parties.

Architecture Goals. The architecture should enable the provider P to get access to the global fee: $A \vdash Has_P^{all}(Fee)$. However, he should not be able to get access to the individual consumptions $Cons_t$ or to the intermediate variables x and y since they are the results of easily inversible functions (typically F is a mapping and S the identity): $A \vdash Has_P^{none}(Cons) \wedge Has_P^{none}(x) \wedge Has_P^{none}(y)$. Moreover, he should be convinced that the value provided for *Fee* is actually correct: $A \vdash K_P(Fee = \odot + (y) \wedge y_t = F(x_t) \wedge x_t = S(Cons_t))$.

Architecture Design. The design of an architecture meeting the above goals is described Figure 1. A strong constraint concerning the metering has to be taken into account from the start: regulators generally require the data to be metered by officially certified and tamper-resistant metrological devices M: $Has_M(Cons)$, $Compute_M(x_t = S(Cons_t))$, and $Attest_M(x_t = S(Cons_t))$.

One option for the computation of the fee is to have it performed by the meter: $Compute_M(Fee = \odot + (y))$ and $Compute_M(y_t = F(x_t))$. The result of this computation can then be sent to the provider along with the corresponding attestation and the metering attestation through a $Receive_{P,M}(\{Att\}, \{Fee\})$ primitive. Another architectural primitive $Verif_P^{Attest}(Att)$ should be added to

convince the provider of the correctness of the computation (considering that the provider trusts the meter $Trust_{P,M}$).

Finally, the dependance relations have to be defined to model the computational power of the components P and M (they both have the same here for the sake of simplicity, noted Dep_i for $i \in \{P, M\}$). The relations are such that $(Fee, \{y_t\}) \in Dep_i$, $(y_t, \{x_t\}) \in Dep_i$, $(x_t, \{y_t\}) \in Dep_i$, $(x_t, \{Cons_t\}) \in Dep_i$, and $(Cons_t, \{x_t\}) \in Dep_i$ (only the summation is not inversible here and we have $(y_t, \{Fee\}) \notin Dep_i$).

Application of the Axiomatics. Rules (H2) and (H6) allow us to prove respectively that the provider gets a value for the global fee since it receives it from the meter and that the consumption and the values of the intermediate variables x and y are not disclosed. Applications of rules (K5) and (K$\wedge$) prove that the correctness of the global fee is ensured thanks to the attestations and the trust relation between the provider and the meter. As expected, (H2) and (H3) prove that the meter has an access to the consumption data.

The solution chosen here for the sake of conciseness describes heavy meters performing the billing computations (which is generally not the case). Moreover, there is a direct link between the meter and the provider: the customer has to trust the meter not to disclose too much data to the provider. This issue could be solved by adding a proxy under the control of the customer which would filter the communications between the provider and the meter. Other options for smart metering such as [29] can be expressed in the same framework but space considerations prevent us from presenting them here.

5 Related Work

This paper stands at the crossroads of three different areas: engineering privacy by design, software architectures and protocols, and epistemic logics.

Several authors [16,19,22,26,32] have already pointed out the complexity of "privacy engineering" as well as the "richness of the data space"[16] calling for the development of more general and systematic methodologies for privacy by design. As far as privacy mechanisms are concerned, [19,23] points out the complexity of their implementation and the large number of options that designers have to face. To address this issue and favor the adoption of these tools, [19] proposes a number of guidelines for the design of compilers for secure computation and zero-knowledge proofs whereas [13] provides a language and a compiler to perform computations on private data by synthesising zero-knowledge protocols. In a different context (designing information systems for the cloud), [24] also proposes implementation techniques to make it easier for developers to take into account privacy and security requirements.

Software architectures have been an active research topic for several decades [31] but they are usually defined using purely graphical, informal means or within semi-formal frameworks. Dedicated languages have been proposed to specify privacy properties [3,5,21,34] but the policies expressed in these languages are usu-

ally more fine-grained than the properties considered here because they are not intended to be used at the architectural level. Similarly, process calculi such as the applied π-calculus [30] have been applied to define privacy protocols [8]. Because process calculi are general frameworks to model concurrent systems, they are more powerful than dedicated frameworks. The downside is that protocols in these languages are expressed at a lower level and the tasks of specifying a protocol and its expected properties are more complex [25,27,6]. Again, the main departure of the approach advocated in this paper with respect to this trend of work is that we reason at the level of architectures, providing ways to express properties without entering into the details of specific protocols that we assume perfect. The work presented here is a follow-up of [1] which advocates an approach based on formal models of privacy architectures. The framework introduced in [22] includes an inference system to reason about the implementation of a "detectability property" similar to the integrity property considered here. This framework makes it possible to prove that, in a given architecture, an actor "A" can detect potential errors (or frauds) in the computation of a variable "X". The logical framework presented here can be seen as a generalisation of [22] which does not include a logic for defining privacy and integrity properties.

Epistemic logics have been extensively studied [12]. A difficulty in this kind of framework in a context where hiding functions are used is the problem known as "logical omniscience". Several ways to solve this difficulty have been proposed [28,17,7]. Other works such as [15] also rely on deontic logics and focus on the expression of policies and how they relate to database security or distributed systems.

6 Directions for Further Work

The framework presented in this paper can be used to express in a formal way the main architectural choices in the design of a system and to reason about them. It also makes it possible to compare different options, based on the properties that they comply with, which is of prime importance when privacy requirements have to be reconciled with other, apparently conflicting requirements.

As stated above, the framework described here does not cover the full development cycle: ongoing work addresses the mapping from the architecture level to the protocol level to ensure that a given implementation, abstracted as an applied π-calculus protocol [30], is consistent with an architecture. Work is also ongoing to integrate this formal framework into a more user-friendly, graphical, design environment integrating a pre-defined design strategy. This strategy, which is implemented as a succession of question-answer iterations, allows the designer to find his way among all possible design options based on key decision factors such as the trust assumptions between entities. The resulting architectures can then be checked using the formal framework described here.

In this paper, we have focused on data minimisation and it should be clear that the framework presented here does not address other privacy requirements such as the purpose limitation or the deletion obligation. Indeed, privacy is a

multi-faceted notion that cannot be entirely captured within a single formal framework. Another limitation of the approach is that it must be possible to define the service (or "purpose") as the result of a functional expression (e.g. the computation of a fee in electronic toll pricing or smart metering). 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). Last but not least, in this paper, follow a "logical" (or qualitative) approach, as opposed to a quantitative approach to privacy and we do not consider the use of auxiliary information. An avenue for further research in this area would be to study the integration of quantitative measures of privacy (such as differential privacy [10]) into the framework.

Acknowledgement. This work was partially funded by the European project PRIPARE/FP7-ICT-2013-1.5, the ANR project BIOPRIV, and the Inria Project Lab CAPPRIS.

References

1. Antignac, T., Le Métayer, D.: Privacy by design: From technologies to architectures. In: Privacy Technologies and Policy, LNCS, vol. 8450, pp. 1–17. Springer (2014)
2. Balasch, J., Rial, A., Troncoso, C., Geuens, C.: PrETP: Privacy-Preserving electronic toll pricing. In: Proc. of the 19th USENIX Security Symp. pp. 63–78. USA (2010)
3. Barth, A., Datta, A., Mitchell, J., Nissenbaum, H.: Privacy and contextual integrity: framework and applications. In: 2006 IEEE Symposium on Security and Privacy. pp. 15–198 (2006)
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. SEI series in Software Engineering, Addison-Wesley, 3rd edn. (2012)
5. Becker, M.Y., Malkis, A., Bussard, L.: A Practical Generic Privacy Language, LNCS, vol. 6503, pp. 125–139. Springer (2011)
6. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Trans. Comput. Syst. 8, 18–36 (1990)
7. Cohen, M., Dam, M.: A complete axiomatization of knowledge and cryptography. In: 22nd Annual IEEE Symp. on Logic in Comp. Science, 2007. pp. 77–88 (2007)
8. Delaune, S., Kremer, S., Ryan, M.D.: Verifying privacy-type properties of electronic voting protocols: A taster. In: Towards Trustworthy Elections – New Directions in Electronic Voting, LNCS, vol. 6000, pp. 289–309. Springer (2010)
9. Diaz, C., Kosta, E., Dekeyser, H., Kohlweiss, M., Girma, N.: Privacy preserving electronic petitions. Identity in the Information Society 1(1), 203–209 (2009)
10. Dwork, C.: Differential privacy. In: Automata, Languages and Programming, vol. 4052, pp. 1–12. Springer, Berlin (2006)
11. European Parliament: European parliament legislative resolution of 12 march 2014 on the proposal for a regulation of the european parliament and of the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data. General Data Protection Regulation, Ordinary legislative procedure: first reading (March 2014)

12. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.: Reasoning About Knowledge. MIT Press (2004)
13. Fournet, C., Kohlweiss, M., Danezis, G., Luo, Z.: Zql: A compiler for privacy-preserving data processing. In: Proc. of the 22Nd USENIX Conference on Security. pp. 163–178. USA (2013)
14. Garcia, F., Jacobs, B.: Privacy-friendly energy-metering via homomorphic encryption. In: Security and Trust Manag., LNCS, vol. 6710, pp. 226–238. Springer (2011)
15. Glasgow, J., MacEwen, G., Panangaden, P.: A logic for reasoning about security. In: Proc. of the 3rd Computer Security Foundations Workshop. pp. 2–13 (1990)
16. Gürses, S., Troncoso, C., Diaz, C.: Engineering Privacy by Design. Presented at the Computers, Privacy & Data Protection conf. (2011)
17. Halpern, J.Y., Pucella, R.: Dealing with logical omniscience. In: Proc. of the 11th Conf. on Th. Aspects of Rationality and Knowl. pp. 169–176. ACM, USA (2007)
18. de Jonge, W., Jacobs, B.: Privacy-Friendly electronic traffic pricing via commits. In: Formal Aspects in Security and Trust, vol. 5491, pp. 143–161. Springer, Berlin (2008)
19. Kerschbaum, F.: Privacy-preserving computation. In: Privacy Technologies and Policy, LNCS, vol. 8319, pp. 41–54. Springer (2014)
20. Krumm, J.: A survey of computational location privacy. Personal and Ubiquitous Computing 13(6), 391–399 (2009)
21. Le Métayer, D.: A Formal Privacy Management Framework. In: Formal Aspects in Security and Trust. LNCS, vol. 5491, pp. 162–176. Springer, Spain (2009)
22. Le Métayer, D.: Privacy by design: A formal framework for the analysis of architectural choices. In: Proc. of the 3rd ACM Conference on Data and Application Security and Privacy. pp. 95–104. ACM, USA (2013)
23. Maffei, M., Pecina, K., Reinert, M.: Security and privacy by declarative design. In: IEEE 26th Computer Security Foundations Symposium. pp. 81–96 (2013)
24. Manousakis, V., Kalloniatis, C., Kavakli, E., Gritzalis, S.: Privacy in the cloud: Bridging the gap between design and implementation. In: Advanced Information Systems Engineering Workshops, Lecture Notes in Business Information Processing, vol. 148, pp. 455–465. Springer (2013)
25. Meadows, C.: Formal methods for cryptographic protocol analysis: emerging issues and trends. IEEE Journal on Selected Areas in Comm. 21(1), 44 – 54 (2003)
26. Mulligan, D.K., King, J.: Bridging the gap between privacy and design. University of Pennsylvania Journal of Constitutional Law 14(4), 989–1034 (2012)
27. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6(1-2), 85–128 (1998)
28. Pucella, R.: Deductive algorithmic knowledge. CoRR cs.AI/0405038 (2004)
29. Rial, A., Danezis, G.: Privacy-Preserving smart metering. Technical report MSR-TR-2010-150, Microsoft Research (2010)
30. Ryan, M.D., Smyth, B.: Formal Models and Techniques for Analyzing Security Protocols, Cryptology and Information Security Series, vol. 5, chap. Applied pi calculus, pp. 112–142. IOS Press (2011)
31. Shaw, M., Clements, P.: The golden age of software architecture. IEEE Softw. 23(2), 31–39 (2006)
32. Spiekermann, S., Cranor, L.F.: Engineering privacy. IEEE Transactions on Software Engineering 35(1), 67–82 (2009)
33. Tschantz, M.C., Wing, J.M.: Formal methods for privacy. In: Proc. of Formal Methods, LNCS, vol. 5850, pp. 1–15. Springer (2009)
34. Yu, T., Li, N., Antón, A.I.: A formal semantics for P3P. In: Proc. of the 2004 Workshop on Secure Web Service. pp. 1–8. SWS '04, ACM, USA (2004)