

# Privacy by Design: From Technologies to Architectures (Position Paper)\*

Thibaud Antignac and Daniel Le Métayer

Inria, Université de Lyon

`thibaud.antignac@inria.fr/daniel.le-metayer@inria.fr`

**Abstract.** Existing work on privacy by design mostly focus on technologies rather than methodologies and on components rather than architectures. In this paper, we advocate the idea that privacy by design should also be addressed at the architectural level and be associated with suitable methodologies. Among other benefits, architectural descriptions enable a more systematic exploration of the design space. In addition, because privacy is intrinsically a complex notion that can be in tension with other requirements, we believe that formal methods should play a key role in this area. After presenting our position, we provide some hints on how our approach can turn into practice based on ongoing work on a privacy by design environment.

## 1 Introduction

Privacy by design is often held up as a necessary step to improve privacy protection in the digital society [32,50]. It will even become a legal obligation in the European Community<sup>1</sup> if the current draft of Data Protection Regulation [15] eventually gets adopted. The fact that future regulatory frameworks promote or impose privacy by design is definitely a positive step but their adoption will take time, they are unlikely to be applicable in all regions of the world and the interpretation of the principle itself may vary greatly<sup>2</sup> among jurisdictions. Therefore, the possible adoption of privacy by design laws should obviously not be seen as the end of the story: the extent to which a true privacy by design will actually turn to reality will also depend on other factors such as social demand, economic conditions and of course the availability of technical solutions. Even though all these dimensions are essential and all possible means should be used

---

\* The final publication is available at link.springer.com ([http://link.springer.com/chapter/10.1007/978-3-319-06749-0\\_1](http://link.springer.com/chapter/10.1007/978-3-319-06749-0_1)).

<sup>1</sup> And also “a prerequisite for public procurement tenders according to the Directive of the European Parliament and of the Council on public procurement as well as according to the Directive of the European Parliament and of the Council on procurement by entities operating in the water, energy, transport and postal services sector.”

<sup>2</sup> And it is obviously out of question to define it very precisely in a legal document.

to foster the adoption of privacy by design, we choose to focus on the technical issues in this paper.

As discussed by several authors [12,21,28,19], a range of privacy enhancing technologies (PETs) are now available, which can provide strong privacy guarantees in a variety of contexts. Even if more research on PETs will always be needed, a question that arises at this stage is how to favor the uptake of these technologies. The position that we want to promote and bring forward for discussion in this paper is threefold:

1. Existing work in this area mostly focus on technologies rather than methodologies and on components rather than architectures. We advocate the idea that privacy by design should also be addressed at the architectural level and be associated with suitable methodologies. Among other benefits, architectural descriptions enable a more systematic exploration of the design space.
2. Because privacy is intrinsically a complex notion, which, in addition, often turns out to be (or seems to be) in tension with other requirements, formal methods should play a key role in this area. Among other benefits, formal descriptions make it possible to define precisely the concepts at hand and the guarantees provided by a solution, to reason about the design choices and ultimately to justify them rigorously, hence contributing also to accountability.
3. Because designers are generally not experts in formal methods, dedicated frameworks and user-friendly interfaces should be designed to hide the complexities of the models and make them usable in this context. Such frameworks should allow designers to express their requirements and to interact with the system to refine or adapt them until a satisfactory architecture is obtained.

The paper is organized as follows: We present our position and discuss it in more detail in Section 2. In order to illustrate this position, we proceed in Section 3 with an example of formal privacy logic which is applied in Section 4 to the design of smart metering systems. Section 5 discusses related work and Section 6 outlines directions for further research.

## 2 Position

A wide array of techniques have been proposed or applied to the definition of new PETs during the last decades [11,18,28,52], including zero-knowledge proofs, secure multi-party computation, homomorphic encryption, commitments, private information retrieval (PIR), anonymous credentials, anonymous communication channels, or trusted platform modules (TPM). These PETs have been used to provide strong privacy guarantees in a variety of contexts such as smart metering [17,33,53], electronic traffic pricing [4,27,49]), ubiquitous computing [32] or location based services [9,29,30]. Even if more techniques will always be needed to defeat new attacks in this eternal game between cops and robbers, there is

now a need for serious thinking about the conditions for the adoption of existing PETs by industry. This reflection has started with convincing cases for the development of appropriate methodologies [12,21,58] or development tools [28] for privacy by design. In this paper, we pursue this line of thought in arguing that privacy by design should also (and firstly) be considered at the architecture level and should be supported by appropriate reasoning tools relying on formal models.

Let us consider the case for architectures first. Many definitions of architectures have been proposed in the literature. In this paper, we will adopt a definition inspired by [6]<sup>3</sup>: *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, as suggested in Section 3, the components are typically the PETs themselves and the purpose of the architecture is their combination to achieve the privacy requirements of the system. Therefore, an architecture is foremost an abstraction and “this abstraction is essential to taming the complexity of a system – we simply cannot, and do not want to, deal with all the complexity all the time” [6].

Most of the reasons identified in [6] to explain why architectures matter are very relevant for privacy by design:

- The architecture is a carrier of the earliest and hence most fundamental hardest-to-change design decisions: overlooking architectural choices may thus seriously hamper the integration of privacy requirements in the design of a system.
- Architectures channel the creativity of developers, reducing design and system complexity: because they make it possible to abstract away unnecessary details and to focus on critical issues, architectures can help privacy designers reason about privacy requirements and the combination of PETs which could be used to meet them.
- A documented architecture enhances communication among stakeholders: documenting the design choices is especially important in the context of privacy to meet the obligations arising from Privacy Impact Assessments (PIA) and accountability (which are emphasized in the Data Protection Regulation draft [15]).
- An architecture can be created as a transferable, reusable model: architectures can therefore play a key role to increase the reusability of privacy friendly solutions, which could lead to significant cost reductions and better dissemination of knowledge and expertise among developers. Being able to go beyond case-by-case strategies and to factorize efforts is a key step for privacy by design to move from craft to industry.

If the choice is made to work at the architectural level, the next question to be answered is: “how to define, represent and use architectures ?” In prac-

---

<sup>3</sup> This definition is a generalization (to system architectures) of the definition of software architectures proposed in [6].

tice, architectures are often described in a pictorial way, using different kinds of graphs with legends defining the meaning of nodes and vertices, or semi-formal representations such as UML diagrams (class diagrams, use case diagrams, sequence diagrams, communication diagrams, etc.). The second point we want to make here is that, even though such pictorial representations can be very useful (especially when their use is standardized), reasoning about privacy requirements is such a subtle and complex issue that the architecture language used for this purpose must be defined in a formal way. By formal, we mean that the properties of the architectures must be defined in a mathematical logic and reasoning about these properties must be supported by a formal proof or verification system. Reasoning about privacy is complex for different reasons: first, it is a multi-faceted notion stemming from a variety of principles which are not defined very precisely. It is therefore of prime importance to circumscribe the problem, to define precisely the aspects of privacy which are taken into account and how they are interpreted in the design choices. The fact that not all aspects of privacy are susceptible to formalization is not a daunting obstacle to the use of formal methods in this context: the key point is to build appropriate models for the aspects of privacy (such as data minimization) which are prone to formalization and involve complex reasoning. Another source of complexity in this context is the fact that privacy often seems to conflict with other requirements such as guarantees of authenticity or correctness, efficiency, usability, etc. To summarize, formal methods should:

- Make it possible to define precisely the concepts at hand (requirements, assumptions, guarantees, etc.).
- Help designers to explore the design space and to reason about the possible choices. Indeed, privacy by design is often a matter of choice [36]: multiple options are generally available to achieve a given set of functionalities, some of them being privacy friendly, others less, and a major challenge for the designer is to understand all these options, their strengths and weaknesses.
- Provide a documented and rigorous justification of the design choices, which would be a significant contribution to the accountability requirement. Accountability is defined in Article 22 of the current draft of the future Data Protection Regulation [15] as the following obligation for data collectors: “The controller shall adopt appropriate policies and implement appropriate and demonstrable technical and organizational measures to ensure and be able to demonstrate in a transparent manner that the processing of personal data is performed in compliance with this Regulation...”

It should be clear however, that designers are generally not experts in formal methods and appropriate tools and user-friendly interfaces have to be made available to hide the complexities of the models. Such frameworks should allow designers to express their requirements, to get understandable representations (or views) of the architectural options, to navigate in these views, and to refine or adapt them until a satisfactory architecture is obtained.

Considering all these requirements, it should be clear that specific, dedicated formal frameworks have to be devised to support privacy by design. To make the

discussion more concrete, we provide in the following sections some hints about such a framework and its application to a real case study.

### 3 Example of Formal Model

As an illustration of the position advocated in the previous section, we present now a subset of a privacy logic to reason about two key properties: the minimization (of the collection of personal data) and the accuracy (of the computations). Indeed, the tension between data minimization and accuracy is one of the delicate issues to be solved in many systems involving personal data. For example, electronic traffic payment systems [4,27,49] have to guarantee both the correctness of the computation of the fee and the limitation of the collection of location data; smart metering systems [17,33,53] have also to ensure the correct computations of the fees and to limit the collection of consumption data; recommendation systems [42] need to achieve both the accuracy of the recommendations while minimizing the disclosure of personal preferences; electric vehicle charging systems [22] also need to guarantee the accuracy of the bills and the protection of the location information of the vehicles.

#### 3.1 Excerpt of a Privacy Epistemic Logic

Because privacy is closely connected with the notion of knowledge, epistemic logics form an ideal basis to reason about privacy properties. Epistemic logics [16] are a family of modal logics using a knowledge modality usually denoted by  $K_i(\phi)$  to express the fact that agent  $i$  knows the property  $\phi$ . 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” [20]. 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 commitment  $C(v)$  (or hash) of a value  $v$  would also know  $v$  (or the possible values of  $v$ ). This is obviously not the intent in a formal model of privacy where commitments 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 and techniques at hand (e.g. knowledge, zero-knowledge proof, trust, etc.). Let us consider for the sake of illustration the following excerpt of a privacy epistemic logic:

$$\phi ::= \phi_0 \quad (1)$$

$$| \neg \phi \quad (2)$$

$$| \phi \wedge \phi' \quad (3)$$

$$| K_i(\phi_0) \quad (4)$$

$$| X_i(\phi_0) \quad (5)$$

$$\phi_0 ::= \text{receive}_{i,j}(x) \quad (6)$$

$$| \text{receive}_{i,j}(\text{prim}) \quad (7)$$

$$| \text{trust}_{i,j} \quad (8)$$

$$| \text{compute}_i(x = t) \quad (9)$$

$$| \text{check}_i(eq) \quad (10)$$

$$| \text{has}_i(x) \quad (11)$$

$$| \text{prim} \mid p \mid \phi_0 \wedge \phi'_0 \quad (12)$$

$$\text{prim} ::= \text{proof}_{i,j}(p) \mid \text{att} \mid \text{prim} \wedge \text{prim}' \quad (13)$$

$$p ::= \text{att} \mid eq \mid p \wedge p' \quad (14)$$

$$\text{att} ::= \text{attest}_i(eq) \quad (15)$$

$$eq ::= t \text{ rel } t' \quad (16)$$

$$\text{rel} ::= = \mid < \mid > \mid \leq \mid \geq \quad (17)$$

$$t ::= c \mid x \mid F(t_1, \dots, t_n) \quad (18)$$

This logic involves only two modalities: the usual knowledge operator  $K_i$  and the “algorithmic knowledge” [16,51] denoted by  $X_i$ , which represents the knowledge that an agent  $i$  can actually build. Properties  $\phi_0$  include both the basic properties  $p$  on variables of the system and properties used to characterize the architecture itself. More precisely,  $\text{receive}_{i,j}(x)$  means that agent  $j$  can send the variable  $x$  to agent  $i$ ,  $\text{receive}_{i,j}(\text{prim})$  means that agent  $j$  can send the property  $\text{prim}$  to agent  $i$  where  $\text{prim}$  can be a combination of (non interactive) zero-knowledge proofs and attestations. An attestation  $\text{attest}_i(eq)$  is a simple declaration by agent  $i$  that property  $eq$  holds. This declaration is of no use to agent  $j$  unless  $j$  trusts  $i$ , which is expressed by  $\text{trust}_{j,i}$ . The primitive  $\text{proof}_{i,j}(p)$  means that agent  $i$  can make a proof of property  $p$  which can be checked by agent  $j$ <sup>4</sup>. The properties  $\text{compute}_i(x = t)$  and  $\text{check}_i(p)$  are used to express the fact that an agent  $i$  can respectively compute a variable  $x$  defined by the equation  $x = t$  or check a property  $p$ . Symbol  $F$  stands for the available basic operations (e.g. hash, homomorphic hash, encryption, random value generation, arithmetic operations, etc.),  $c$  stands for constants, and  $x$  for variables. Last but not least,

---

<sup>4</sup> In general a proof could be checked by several agents, which could be expressed using a set of agents names as second index.

$\text{has}_i(x)$  is the property that agent  $i$  can get the value of variable  $x$  (which does not, in itself, provide any guarantee about the correctness of this value).

### 3.2 Semantics and Axiomatization

The semantics of this logic can be defined using an augmented Kripke model  $M = (Arch, \pi, \mathcal{D}_1, \dots, \mathcal{D}_n)$  where:

- $Arch$  is a set of possible architectures (generally called worlds) of the system under consideration. In our setting, an architecture is defined as a property  $\phi_0$  which characterizes all the operations available to the agents.
- $\pi$  is an interpretation for the primitives  $prim$  and the relations  $eq$ .
- $\mathcal{D}_1, \dots, \mathcal{D}_n$  are the deductive systems associated with agents  $1, \dots, n$ .

The deductive system associated with an agent  $i$  makes it possible to define the semantics of the  $X_i$  operator (knowledge built by agent  $i$ ). In other words, each agent  $i$  can apply rules  $\triangleright_i$  to derive new knowledge. Typical rules of deductive systems include:

- $\text{receive}_{i,j}(prim) \triangleright_i prim$
- $\text{attest}_j(eq), \text{trust}_{i,j} \triangleright_i eq$
- $\text{proof}_{j,i}(p) \triangleright_i p$
- $\text{check}_i(eq) \triangleright_i p$
- $\text{compute}_i(x = t) \triangleright_i x = t$
- $\text{hash}(x_1) = \text{hash}(x_2) \triangleright_i x_1 = x_2$
- $\text{hhash}(x) = \text{hhash}(x_1) \otimes \text{hhash}(x_2) \triangleright_i x = x_1 + x_2$
- $\text{receive}_{i,j}(x) \triangleright_i \text{has}_i(x)$
- $\text{compute}_i(x = t) \triangleright_i \text{has}_i(x)$
- $\text{has}_i(x_1), \dots, \text{has}_i(x_m), \text{dep}(x, \{x_1, \dots, x_m\}) \triangleright_i \text{has}_i(x)$   
with  $\text{dep}(x, \{x_1, \dots, x_m\})$  a dependence relationship known by  $i$  stating that  $x$  can be derived from  $x_1, \dots, x_m$ .

In order to reason about architectures, we can use an axiomatization in the style of previous work on deductive algorithmic knowledge [16,51]. Typical examples of axioms and inference rules include:

$$\mathbf{TAUT}: \text{All instances of propositional tautologies.} \quad (19)$$

$$\mathbf{MP}: \text{From } \phi \text{ and } \phi \rightarrow \psi \text{ infer } \psi \quad (20)$$

$$\mathbf{Gen}: \text{From } \phi \text{ infer } K_i(\phi) \quad (21)$$

$$\mathbf{K}: K_i(\phi \rightarrow \psi) \rightarrow (K_i(\phi) \rightarrow K_i(\psi)) \quad (22)$$

$$\mathbf{T}: K_i(\phi) \rightarrow \phi \quad (23)$$

$$\mathbf{KC}: K_i(\phi \wedge \psi) \rightarrow (K_i(\phi) \wedge K_i(\psi)) \quad (24)$$

$$\mathbf{XD}: \text{From } X_i(\phi_1), \dots, X_i(\phi_n) \text{ and } \phi_1, \dots, \phi_n \triangleright_i \phi \text{ infer } X_i(\phi) \quad (25)$$

$$\mathbf{XT}: X_i(\phi) \rightarrow \phi \quad (26)$$

$$\mathbf{XC}: X_i(\phi \wedge \psi) \rightarrow (X_i(\phi) \wedge X_i(\psi)) \quad (27)$$

A key difference between the properties of  $K_i$  and  $X_i$  is the lack of counterpart of axioms **Gen** and **K** for  $X_i$ , which makes it possible to avoid the omniscience problem. In contrast, the knowledge built by an agent  $i$  depends on the associated deductive system  $\triangleright_i$  as expressed by rule **XD**. Rules **T** and **XT** express the fact that an agent cannot derive false properties.

The axiomatization outlined in this section forms the basis for an inference algorithm which makes it possible to prove properties of architectures (expressed as  $\phi_0$  properties) as discussed in the next subsection.

### 3.3 Use of the Formal Model

The first functionality of a design environment is to make it possible for the designer to express the requirements that apply to the system and, optionally, the design choices which have already been made (or which are imposed by the environment or the client) as well as the possible trust assumptions.

Formally speaking, the requirements are made of three components:

- Functional requirements: the purpose of the system, which is expressed as a set of equations  $x = t$ .
- Privacy and knowledge requirements: values that should not (respectively should) be known by certain actors, which is expressed by properties  $\neg\text{has}_i(x)$  (respectively  $\text{has}_i(x)$ ).
- Correctness (integrity) requirements: the possibility for certain actors to ensure that certain values are correct, which is expressed as  $X_i(eq)$ .

Generally speaking, other non-functional requirements could also be considered but the combination of privacy and correctness already provides a degree of complexity which is sufficient to illustrate the approach.

The design choices, which add further requirements on the architecture, can be defined as  $\phi_0$  properties. They can express, for example, the fact that communication links are (or are not) available between specific components or certain computations (zero-knowledge, homomorphic hash, etc.) can (or cannot) be performed by certain components.

Several situations are possible when the designer has entered this first batch of information:

1. The requirements may be contradictory, for example because privacy requirements conflict with knowledge or architectural requirements or because architectural requirements themselves are not consistent (operations computed by components which cannot get access to the necessary parameters, checks which cannot be carried out because some values are not available to the component, etc.). The system returns the identified contradictions, which may provide some hints to the designer to modify his initial input.
2. The requirements may be consistent but not precise enough to characterize a unique architecture. In this case, the system can use a library of existing PETs to provide suggestions to the user. The user can then decide to apply a given PET (which is expressed formally by the addition of a new assumption,



e.g.  $\text{receive}_{i,j}(\text{proof}_{j,i}(p))$  for a zero-knowledge proof of property  $p$  sent by agent  $j$  to agent  $i$ .

3. The requirements may be precise enough to specify a unique (and correct architecture).

The first two cases lead to a new iteration of the procedure. In the last case, the designer has obtained a satisfactory architecture (which does not prevent him from performing a new iteration with different assumptions to further explore the design space).

## 4 Example of Application

Let us now illustrate the framework outlined in the previous section with a small smart metering case study. Privacy friendly smart grids and smart metering systems have been studied extensively [17,25,26,53]. Our purpose here is neither to present a new solution nor to provide a comprehensive study of smart-metering systems but to show how a formal framework can help a designer to find his way among the possible options. We focus here on the billing functionality and the potential tensions between the privacy requirements of the clients and the need for the operator to ensure the correctness of the computation of the fee.

Let us assume in the first instance that the system is made of three components: the back-end system of the operator  $o$ , the computer of the user (customer)  $u$  and the meter  $m$ . The requirements for this case study are the following:

- Functional requirements: the purpose of the system is the computation of the fee which is expressed as the equation  $Fee = \sum_{i=1}^n (P(C_i))$  where  $C_i$  are the actual consumption values for the considered period of time ( $i \in [1, n]$ ) and  $P$  is the cost function.
- Privacy and knowledge requirements:  $\neg \text{has}_o(C_i)$  and  $\text{has}_o(Fee)$  respectively, to express the fact that the operator  $o$  should not obtain the individual consumption values  $C_i$  but should get the fee.
- Correctness (integrity) requirements: the operator must be sure that the fee is correct:  $X_o(Fee = \sum_{i=1}^n (P(C_i)))$ .

Let us assume in a first scenario that the designer considered a direct communication link between the meter and the operator, which means that the architecture would include the property  $\text{receive}_{o,m}(C_i)$ . This possibility would obviously conflict with the privacy requirement since  $\text{receive}_{o,m}(C_i) \triangleright_i \text{has}_o(C_i)$ . Two communication links are therefore necessary: from  $m$  to  $u$  and from  $u$  to  $o$ .

The next question is where the computation of  $P$  should take place: generally speaking, this could be at the back-office of the operator, on the meter, or on the computer of the user. Depending on his initial ideas about the architecture and the constraints imposed by the hardware, the designer can either enter directly the appropriate  $\text{compute}_o$ ,  $\text{compute}_m$  or  $\text{compute}_u$  property. Otherwise these options would be suggested in turn by the system.

- The first option turns out to conflict with the privacy requirements (because the operator would need the input values  $C_i$  to compute  $Fee$ ) unless homomorphic encryption can be used to allow the operator to compute  $Fee$  on encrypted values.
- The second option can be further explored if it does not incur unacceptable additional costs on the meters. However, the system would then identify a trust requirement (which may or may not have been foreseen by the designer): the operator must trust the meter ( $trust_{o,m}$ ) because the only information received by the operator would be an attestation of the meter ( $attest_m(Fee = \sum_{i=1}^n (P(C_i)))$ ) and an attestation can turn into a true guarantee only in conjunction with a trust assumption (as expressed by the rule  $attest_j(eq), trust_{i,j} \triangleright_i eq$ ).
- The third option will be more appealing if the role of the meters has to be confined to the delivery of the consumption measures. But this choice leads to another requirement: either the user can be trusted by the operator (which is excluded by assumption) or he has to be able to provide to the operator a proof of correctness of the computation of the fee [53] ( $receive_{o,u}(\text{proof}_{u,o}(Fee = \sum_{i=1}^n (P(C_i))))$ ).

If none of these options is available, further actors need to be considered and involved in the computation of the fee. In general, these actors can either be pairs or trusted third parties. In both cases, further requirements would arise: a secure multi-party computation scheme would be necessary to ensure that the pairs do not learn each other’s consumptions and trust assumptions would be necessary to ensure that computations can be delegated to the third party.

As discussed in Section 2, designers are usually not experts in formal methods. As a result, a design environment should provide alternative modes of interaction hiding the complexities of the formal model. Considering that designers are used to graphical notations, we have chosen to represent the different views of the architecture as annotated graphs. The interested reader can find in Annex 1 the “location views” showing, for the two successful scenarios described above, the architectures obtained at the end of the interaction process.

## 5 Related Work

This position paper stands at the crossroads of at least three different areas: software architectures, formal models for privacy and engineering of privacy by design.

Software architectures have been an active research topic for several decades [54] but they are usually defined using purely graphical, informal means [6] or within semi-formal [8] frameworks. Formal frameworks have been proposed to define software architectures [2,23,34,48] but they are usually based on process algebras or graph grammars and they are not designed to express privacy properties. One exception is the framework introduced in [37] which defines the meaning of the available operations in a (trace-based) operational semantics and

proposes an inference system to derive properties of the architectures. The inference system is applied to the proof of properties related to the use of spot checks in electronic traffic pricing systems. Even though the goal of [37] is to deal with architectures, it remains at a lower level of abstraction than the framework sketched here (because of its operational underpinning, which contrasts with the epistemic logic used here) and can hardly be extended to other privacy mechanisms.

On the other hand, dedicated languages have been proposed to specify privacy properties [3,5,7,24,35,38,45,59] but the policies expressed in these languages are usually 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 pi-calculus [1] have been applied to define privacy protocols [10]. 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. 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.

Notions such as  $k$ -anonymity [39,56],  $l$ -diversity [40] or  $\epsilon$ -differential privacy [13,14] have also been proposed as ways to measure the level of privacy provided by an algorithm. Differential privacy provides strong privacy guarantees independently of the background knowledge of the adversary: the main idea behind  $\epsilon$ -differential privacy is that the presence (or the absence) of an item in a database (such as the record of a particular individual) should not change in a significant way the probability of obtaining a certain answer for a given query. Methods [14,41,43] have been proposed to design algorithms achieving privacy metrics or to verify that a system achieves a given level of privacy [57]. These contributions on privacy metrics are complementary to the work described in this paper. We follow 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.

As far as the engineering of privacy is concerned, several authors [12,21,28,46,55] have already pointed out the complexity of problem as well as the “richness of the data space” [12], calling for the development of more general and systematic methodologies for privacy by design. [21] has used design patterns to define eight privacy strategies<sup>5</sup> called respectively: Minimise, Hide, Separate, Aggregate, Inform, Control, Enforce and Demonstrate. As far as privacy mechanisms are concerned, [28] points out the complexity of their implementation and the large number of options that designers have to face. To address this issue and

---

<sup>5</sup> Strategies are defined as follows in [21]: “A design strategy describes a fundamental approach to achieve a certain design goal. It has certain properties that allow it to be distinguished from other (fundamental) approaches that achieve the same goal.”

favor the adoption of these tools, [28] proposes a number of guidelines for the design of compilers for secure computation and zero-knowledge proofs. In a different context (designing information systems for the cloud), [44] also proposes implementation techniques to make it easier for developers to take into account privacy and security requirements. In the same spirit, [47] proposes a decision support tool based on design patterns to help software engineers to take into account privacy guidelines in the early stage of development.

A recent proposal ([31]) also points out the importance of architectures for privacy by design. [31] proposes a design methodology for privacy (inspired by [6]) based on tactics for privacy quality attributes (such as minimization, enforcement or accountability) and privacy patterns (such as data confinement, isolation or Hippocratic management). The work described in [31] is complementary to the approach presented here: [31] does not consider formal aspects while this paper does not address the tactics for privacy by design.

Finally, even though we decided to focus on one (important) aspect of privacy by design here, namely data minimization, other aspects also deserve more attention, in particular the design of appropriate interfaces to allow users to take more informed decisions about their personal data [46].

## 6 Directions for further work

The basic idea put forward in this paper is that privacy by design should be addressed at the architecture level and this should be done, at least for critical aspects such as minimization, in a formal way. As stated in Section 1, some aspects of privacy (such as proportionality or purpose) may be more difficult to formalize or impossible to formalize entirely. This should not be an obstacle to the use of formal methods for other aspects of privacy. Generally speaking, architectures are made of different views [6]: some of them can be described informally or in a purely pictorial way while others are based on a formal model. In any case, working at the architectural level is a key success factor for privacy by design because architectures carry “the earliest and hence most fundamental hardest-to-change design decisions” [6]. They should also play a key role to support accountability requirements because they can provide evidence that appropriate decisions have been taken and the reasons for taking them.

Obviously the design of an architecture meeting all privacy requirements is not the end of the story: because an architecture is set, by definition, at a fairly high level of abstraction, the remaining task is to use or devise appropriate mechanisms to implement it. In some sense, the architecture defines precise requirements on the techniques (PETs) to be used in the implementation. The approach presented here is therefore complementary to the work done on the improvement of the implementation of privacy mechanisms [28]. It is also complementary to the ongoing work on the conception of new PETs. In practice, a design environment should include a library of available PETs with their associated guarantees (in the privacy logic) and this library should follow the progress of the technologies. One challenge to this respect will be to ensure that the

logical framework is general and versatile enough to allow the description of future mechanisms. We believe that epistemic logics provide a well-established and suitable framework to this aim but this claim has to be confirmed in practice.

As far as the formal framework itself is concerned, we have suggested a “logical” (or qualitative) approach in this paper. 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. This extension would be required to deal with the (numerous) situations in which data cannot be classified into two categories (can or cannot be disclosed) but can be disclosed provided a sufficiently robust sanitization algorithm is applied. Further work on this issue will benefit from existing results [14,41,43] on the design of mechanisms achieving differential privacy.

In this paper we provided some hints on how our approach can turn into practice based on ongoing work on a privacy by design environment. Needless to say, more work has to be done on the HCI front, to improve the interactions with designers. As suggested in Section 3, graphical means can be used to convey the essence of the properties in the logic and the architecture under construction, for example to picture data flows, the locations of the computations and the trust relationships. A key concept to this respect is the notion of view in software architectures. More experience is needed, however, to understand what will be the most useful views for a designer.

Last but not least, another interesting avenue for further research would be to apply this approach to other aspects of privacy (using combinations of formal and semi-formal methods) and to establish a link with the coarse-grain strategies defined in [21] to drive the interactions with the system.

**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 (Collaborative Action on the Protection of Privacy Rights in the Information Society.)

## References

1. M. Abadi, and C. Fournet. Mobile Values, New Names, and Secure Communication. Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 104–115, 2001.
2. R. Allen, and D. Garlan. Formalizing Architectural Connection. Proc. 16th Int’l Conf. Software Eng., pages 71–80, May 1994.
3. M. Backes, M. Dürmuth, and G. Karjoth. Unification in privacy policy evaluation - translating EPAL into Prolog. In *POLICY*, pages 185–188, 2004.
4. 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.
5. 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.

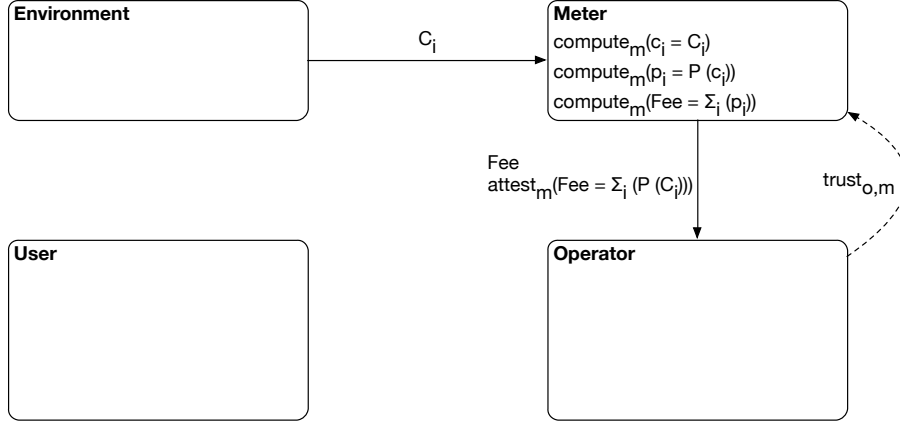
6. L. Bass, P. Clements, and R. Kazman. Software architecture in practice (3d edition). SEI Series in Software Engineering, Addison Wesley, 2013.
7. M. Y. Becker, A. Malkis, and L. Bussard. A practical generic privacy language. In *ICISS*, pages 125–139, 2010.
8. G. Booch, I. Jacobson, and J. Rumbaugh. The Unified Modeling Language Reference Manual (2nd Edition) Addison Wesley Professional, 2004.
9. M. L. Damiani, E. Bertino, and C. Silvestri. The probe framework for the personalized cloaking of private locations. *Transactions on Data Privacy* 3(2), pages 123–148, 2010.
10. S. Delaune, S. Kremer, and M. D. Ryan. Verifying Privacy-type Properties of Electronic Voting Protocols. *Journal of Computer Security* 17(4), pages 435–487, 2009.
11. Y. Deswarte and C. A. Melchor. Current and future privacy enhancing technologies for the internet. *Annals of Telecommunications* 61(3), pages 399–417, 2006.
12. S. F. Gürses, C. Troncoso, and C. Diaz. Engineering privacy by design. In *Computers, Privacy & Data Protection*, (2011).
13. C. Dwork. Differential privacy. In *ICALP (2)*, pages 1–12, 2006.
14. C. Dwork. A firm foundation for private data analysis. *Commun. ACM* 54(1), pages 86–95, 2011.
15. E.C. European Commission. 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). inofficial consolidated version after LIBE Committee vote provided by the rapporteur, 22 October 2013.
16. R. Fagin, J. Y. Halpern, Y. Moses, M. Vardi. Reasoning About Knowledge. A Bradford Book; 1st MIT Press Paperback Ed edition (January 9, 2004).
17. 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.
18. 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.
19. M. Hafiz. A Pattern Language for Developing Privacy Enhancing Technologies. *Software — Practice and Experience* 43(7), pages 769–787, 2013.
20. J. Y. Halpern, R. Pucella. Dealing with logical omniscience: Expressiveness and pragmatics. *Artif. Intell.* 175(1), pages 220–235, 2011.
21. J.-H. Hoepman. Privacy Design Strategies. CoRR 2013.
22. C. Höfer, J. Petit, R. Schmidt, and F. Kargl. 2013. POPCORN: privacy-preserving charging for e-mobility. In *Proceedings of the 2013 ACM workshop on Security, privacy & dependability for cyber vehicles (CyCAR '13)*. ACM, New York, NY, USA, pages 37–48, 2013.
23. P. Inverardi, and A. Wolf. Formal specification and analysis of software architectures using the chemical abstract machine model. *IEEE Transactions on Software Engineering* 21(4), pages 373–386. Special Issue on Software Architectures, 1995.
24. 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.
25. M. Jawurek, M. Johns, F. Kerschbaum. Plug-In Privacy for Smart Metering Billing. *Privacy Enhancing Technologies Symposium (PETS'11)*, pages 192–210, 2011.
26. M. Jawurek, F. Kerschbaum, and G. Danezis. Privacy Technologies for Smart Grids - A Survey of Options. MSR-TR-2012-119. Nov. 2012.

27. W. D. Jonge and B. Jacobs. Privacy-friendly electronic traffic pricing via commits. In *Proceedings of the Workshop of Formal Aspects of Security and Trust*, pages 132–137. Springer, LNCS 5491, 2009.
28. F. Kerschbaum. Privacy-Preserving Computation (Position Paper). Annual Privacy Forum (APF'12), Cyprus, 2012.
29. 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: pages 139–146, 2008.
30. J. Krumm. A survey of computational location privacy. *Pers Ubiquit Comput*, 13, pages 391–399, 2008.
31. A. Kung. PEARs: Privacy Enhancing ARchitectures. Annual Privacy Forum (APF'14), Greece, 2014.
32. M. Langheinrich. Privacy by design - principles of privacy aware ubiquitous systems. In *Proceedings of the Ubicomp Conference*, Springer, LNCS 2201, pages 273–291, 2001.
33. M. LeMay, G. Gross, C. A. Gunter, and S. Garg. Unified architecture for large-scale attested metering. In *HICSS*, pages 115–124, 2007.
34. D. Le Métayer. Software Architecture Styles As Graph Grammars. ACM SIGSOFT Software Eng. Notes, Nov. 1996.
35. D. Le Métayer. A formal privacy management framework. In *FAST (Formal Aspects of Security and Trust)*, Springer, LNCS 5491, pages 161–176, 2009.
36. D. Le Métayer. Privacy by design: a matter of choice. In *Data Protection in a Profiled World*, Springer Verlag, pages 323–334, 2010.
37. D. Le Métayer. Privacy by design: a formal framework for the analysis of architectural choices. CODASPY'13, pages 95–104, 2013.
38. N. Li, T. Yu, and A. I. Antón. A semantics based approach to privacy languages. *Comput. Syst. Sci. Eng.*, 21(5), 2006.
39. N. Li, W. H. Qardaji, and D. Su. Provably private data anonymization: Or, k-anonymity meets differential privacy. *CoRR*, abs/1101.2604, 2011.
40. A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data* 1(1) Article 3, March 2007.
41. F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. *Commun. ACM*, 53(9), pages 89–97, 2010.
42. F. McSherry and I. Mironov. 2009. Differentially private recommender systems: building privacy into the net. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '09)*. ACM, New York, NY, USA, pages 627–636, 2009.
43. F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103, 2007.
44. V. Manousakis, C. Kalloniatis, E. Kavakli, S. Gritzalis. Privacy in the Cloud: Bridging the Gap between Design and Implementation. *Proceedings of the WISSE 2013 3rd International Workshop on Information Systems Security Engineering*, in conjunction with CAiSE 2013 25th International Conference on Advanced Information Systems Engineering, X. Franch and P. Soffer (Eds.), Valencia, Spain, Springer Lecture Notes in Business Information Processing 148, Jun. 2013.
45. 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.
46. D. K. Mulligan, and J. King. Bridging the Gap between Privacy and Design. *University of Pennsylvania Journal of Constitutional Law*, (14)4, 2012.

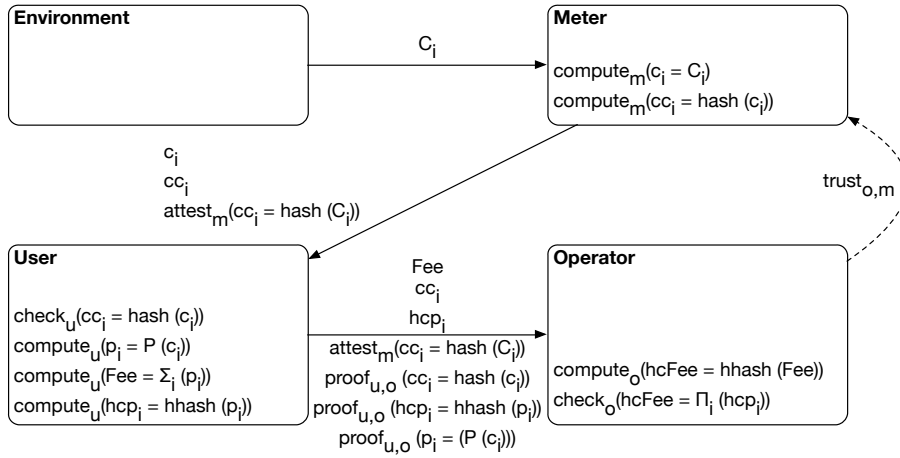
47. S. Pearson and A. Benameur. A Decision Support System for Design for Privacy. Privacy and Identity, IFIP AICT 352, pp. 283–296, 2011.
48. D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. ACM SIGSOFT Software Eng. Notes, Oct. 1992.
49. R. A. Popa, H. Balakrishnan, and A. J. Blumberg. Vpriv: Protecting privacy in location-based vehicular services. In *USENIX Security Symposium*, pages 335–350, 2009.
50. Y. Pouillet. 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.
51. R. Pucella. Deductive Algorithmic Knowledge. *Journal of Logic and Computation* 16 (2), pages 287–309, 2006.
52. A. Rezgui, A. Bouguettaya, and M. Y. Eltoweissy. Privacy on the web: facts, challenges, and solutions. *IEEE Security and Privacy*, pages 40–49, 2003.
53. 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.
54. M. Shaw, and P. Clements. The Golden Age of Software Architecture: A Comprehensive Survey. Research Report CMU-ISRI-06-101. Carnegie Mellon University, 2006.
55. S. Spiekermann and L. F. Cranor. Engineering Privacy. *IEEE Transactions on Software Engineering* 35(1), 2009.
56. L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5), pages 557–570, 2002.
57. M. C. Tschantz, D. K. Kaynar, and A. Datta. Formal verification of differential privacy for interactive systems. *CoRR*, abs/1101.2819, 2011.
58. M. C. Tschantz and J. M. Wing. Formal methods for privacy. In *Proceedings of the 2nd World Congress on Formal Methods (FM’09)*, pages 1–15, 2009.
59. T. Yu, N. Li, and A. I. Antón. A formal semantics for P3P. In *In Proceedings of the 2004 workshop on Secure web service (SWS ’04)*, pages 1–8, 2004.



## Annex 1



**Fig. 1.** Option 2 of Section 4: The meter computes the fee and has just to be trusted by the operator.  $C_i$  are the actual consumptions and  $c_i$  the values actually used by the meter.



**Fig. 2.** Option 3 of Section 4: The user computes the fee and the operator has to trust the meter (for using the right value  $c_i$ ).  $\text{hhash}$  is a homomorphic hash function which allows the operator to check that the hash of the global fee **Fee** is consistent with the hashes  $hcp_i$  of individual fees  $p_i$ . This architecture is an abstraction of the solution proposed by [53].