Trust Driven Strategies for Privacy by Design

Thibaud Antignac(✉) and Daniel Le Métayer

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

Abstract. In this paper, we describe a multi-step approach to privacy by design. The main design step is the choice of the types of trust that can be accepted by the stakeholders, which is a key driver for the construction of an acceptable architecture. Architectures can be initially defined in a purely informal way and then mapped into a formal dedicated model. A tool integrating the approach can be used by designers to build and verify architectures. We apply the approach to a case study, an electronic toll pricing system, and show how different solutions can be suggested to the designer depending on different trust assumptions.

1 Introduction

The general philosophy of privacy by design is that privacy should not be treated as an afterthought but rather as a first-class requirement in the design of IT systems. In other words, designers should have privacy in mind from the start when they define the features and architecture of a system. Privacy by design will become a legal obligation in the European Union if the current draft of the General Data Protection Regulation (GDPR) [8] eventually gets adopted. However, it is one thing to impose by law the adoption of privacy by design, quite another to define precisely what it is intended to mean and to ensure that it is put into practice. In fact, privacy by design is a particularly challenging endeavour, for plenty of reasons:

- First, privacy itself is a very general principle, but it is also a very subjective notion, which evolves over time and depends very much on the cultural and technological context. Therefore, the first task in the perspective of privacy by design is to define precisely the privacy requirements of the system.
- Privacy is often (or often seems to be) in tension with other requirements, for example functional requirements, ease of use, performances or accountability.
- A wide array of privacy enhancing technologies (PETs) have been proposed during the last decades (including zero-knowledge proofs, secure multi-party computation, homomorphic encryption, etc.) Each of them 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.

C.D. Jensen et al. (Eds.): IFIPTM 2015, IFIP AICT 454, pp. 60–75, 2015.
DOI: 10.1007/978-3-319-18491-3_5

In this context, a major challenge for the designer is to understand all the possible options and their strengths and weaknesses. On the basis of the above, we believe that the most urgent needs in this area are:

1. The availability of strategies for the search of solutions based on the different requirements of the system and the available PETs.
2. The possibility to express these solutions in a formal framework and to reason about their properties.
3. The existence of a link between the strategies and the formal framework to capitalise on the knowledge gained in the design phase to facilitate the specification and verification phases.

The last item is of prime importance especially because designers should not be expected to be experts in formal methods (or even to be ready to be confronted with them at all). Therefore, the output of the design phase, which is conducted in a non-formal environment, should be translated automatically in the formal framework. In addition, this translation should take advantage of the knowledge conveyed by the designer during the first phase because this knowledge can be exploited to set the assumptions and prove the required properties. To this respect, a key decision which has to be made during the design phase is the choice of the trust relationships between the parties: this choice is both a driving factor in the selection of architectural options and a critical assumption for the proof of properties of the solution.

In this paper, we propose an approach for the reasoned construction of architectures and we illustrate it with one aspect of privacy which is often called data minimisation. We describe the overall approach and methodology in Sect. 2 and outline *CAPRIV*, our computer assisted privacy engineering tool in Sect. 3. In Sect. 4, we apply the approach to a case study, an electronic toll pricing system. Section 5 discusses related work and Sect. 6 outlines directions for further research.

2 Trust Driven Strategies

A wide range of PETs are now available, which can provide strong privacy guarantees in a variety of contexts [6,11,17,27]. However, the take-up of privacy by design in the industry is still rather limited. This situation is partly due to legal and economic reasons, but one must also admit that no general methodology is available to help designers choosing among existing techniques and integrating them in a consistent way to meet a set of privacy requirements. The next challenge in this area is therefore to go beyond individual cases and to establish sound foundations and methodologies for privacy by design [7,34]. We advocate the idea that privacy by design should first be addressed at the architectural level because the abstraction level provided by architectures makes it easier to express the key design choices and to explore in a more systematic way the design space. In this section, we first set the stage and define the type of system and requirements considered here (Subsect. 2.1) before defining briefly our notion of

architectures (Subsect. 2.2) and describing the overall strategies and criteria used for the construction of a privacy compliant architecture (Subsect. 2.3).

2.1 Context: Data Minimisation and Integrity

Data minimisation is one of the key principles of most privacy guidelines and regulations. 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[1].

In practice, however, apart from cases in which the purpose can be achieved without the collection of any personal data at all, there is usually no real notion of minimality in a mathematical sense of the term. This is the case for different reasons: first, the purpose itself cannot always be defined formally and so is subject to interpretation; for example, services can sometimes be improved through the disclosure of additional personal data[2]. In addition, different requirements (functional or non functional) usually have to be met at the same time and these requirements can be in tension or conflicting with data minimisation. One common requirement which has to be taken into account is what we call "integrity" in the sequel, to describe 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. For example, electronic toll pricing systems [16,26,33] have to guarantee both the correctness of the computation of the fee and the limitation of the collection of location data; smart metering systems [10,20,28] also have to ensure the correct computations of the fees and the supply-demand balance of the network while limiting the collection of consumption data, etc.

The best that can be done to cope with these requirements is therefore to be able to describe them in a uniform framework and to reason about their relationships, to select the architecture that meets them all if possible or to decide whether certain assumptions could be changed (for example by introducing a trusted third party) or whether certain requirements can be relaxed.

In this paper we illustrate our approach with the two types of requirements discussed here: on the one hand, minimisation as a requirement of the data subject and, on the other hand, integrity as a requirement of both the service provider and the data subject who need guarantees about the result of a computation involving personal data. The meaning of these requirements depends on the purpose of the data collection, which is equated to the expected functionality of the system. In the sequel, we assume that this functionality Ω is expressed as the computation of a set of equations[3] as described in Table 1.

[1] See for example Article 5(c) of the draft of the GDPR [8].

[2] Indeed, improving the user's experience through personalisation is a common excuse for justifying the collection of large amounts of data.

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

Table 1. Functionality language.

$$\Omega ::= \{X = T\}$$
$$T ::= X \mid \mathrm{F}(X_1, \ldots, X_n) \mid \odot \mathrm{F}(X)$$

The notation $\{Z\}$ is used to define a set of elements of category Z and T defines terms over (array or simple) variables $X \in \mathit{Var}$. Function applications are denoted $\mathrm{F}(X_1, \ldots, X_n)$ with $\mathrm{F} \in \mathit{Fun}$ and iterative function applications are denoted by $\odot \mathrm{F}(X)$ with F the function iteratively applied to the elements of the array X (e.g. sum of the elements of X if F is equal to $+$).

As an example, electronic toll pricing allows drivers to be charged depending on their actual behavior (mileage, time, ...). The global fee (*fee*) due to the toll service provider (*SP*) at the end of the billing period is based on the use of the road infrastructures modeled as trajectory location parts (denoted by an array *loc* metered by the on-board unit (*OBU*) of the driver's vehicle over periods of time n). The service $\mathit{fee} = \sum_n (\mathrm{F}(\mathit{loc}_n))$ (where F stands for a pricing function) is expressed in our language as $\Omega = \{\mathit{fee} = \odot + (p), p = \mathrm{F}(\mathit{loc})\}$ (with p an intermediate array variable standing for the prices corresponding to *loc*).

Privacy requirements are used to express the expected properties of an architecture in terms of confidentiality and integrity. The syntax is defined in Table 2.

Table 2. Privacy requirements language.

$$\phi ::= \mathit{Has}_i^{all}(X) \mid \mathit{Has}_i^{none}(X) \mid \mathit{Has}_i^{one}(X) \mid K_i(\mathit{Eq}) \mid B_i(\mathit{Eq}) \mid \phi_1 \wedge \phi_2$$
$$\mathit{Eq} ::= T_1 \ \mathit{Rel} \ T_2 \qquad \mathit{Rel} ::= = \mid < \mid > \mid \leq \mid \geq$$

The subscript i stands for a component index and Eq are equations over terms T and operators *Rel*. $\mathit{Has}_i^{all}(X)$ expresses the fact that component C_i can obtain the value of X (or all the values of X in the case of an array). $\mathit{Has}_i^{none}(X)$ is the confidentiality property stating that C_i cannot obtain the value of X (or any element of X in the case of an array). Finally, $\mathit{Has}_i^{one}(X)$ expresses the fact that component C_i can obtain the value of at most one element of the array X. It should be noted that $\mathit{Has}_i^{all}(X)$, $\mathit{Has}_i^{none}(X)$, and $\mathit{Has}_i^{one}(X)$ properties only inform on the fact that C_i can get values for the variables but they do not bring any guarantee about the correctness of these values. Such integrity requirements can be expressed using the $K_i(Eq)$ and $B_i(Eq)$ properties for knwoledge and belief respectively. $K_i(Eq)$ means that component C_i can establish with certainty the truthfulness of Eq while $B_i(Eq)$ expresses the fact

that C_i can establish with certainty or with a limited (and reasonable) amount of uncertainty this truthfulness (C_i may detect its falsehood if he can test this truthfulness for a sample).

In the example, the provider can obtain the value of the global fee ($Has^{all}_{SP}(fee)$). Moreover, we assume that the driver does not want the provider to obtain the value of its locations (loc) or intermediate prices (p). However, the reception by the provider of a sample of these values is allowed if needed for a posteriori integrity verification ($Has^{one}_{SP}(loc)$ and $Has^{one}_{SP}(p)$). Moreover, the architecture must ensure that the provider can test the correction of the computation of the prices ($B_{SP}(p = \mathrm{F}(loc))$) and establish the correction of the aggregation of these prices ($(K_{SP}(fee = \odot + (p)))$[4].

2.2 Architectures

Many definitions of architectures have been proposed in the literature. In this paper, we adopt a definition inspired by [3][5]: *The architecture of a system is the set of structures needed to reason about the system, which comprises software and hardware elements, relations among them and properties of both.* In the context of privacy, the components are typically the PETs themselves and the purpose of the architecture is their combination to achieve the privacy requirements of the system. As suggested above, an architecture is foremost an abstraction of a system and, as argued in [3], "this abstraction is essential to taming the complexity of a system". Following this principle, a set of components C_i is 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 [1] as detailed in Table 3.

Table 3. Privacy architecture language.

$A ::= \{R\}$		
$R ::= Has_i(X)$	$\mid Receive_{i,j}(\{S\},\{X\})$	
$\mid Compute_i(X = T)$	$\mid Check_i(\{Eq\})$	
$\mid Verif_i^{Proof}(Pro)$	$\mid Verif_i^{Attest}(Att)$	
$\mid Spotcheck_{i,j}(X,\{Eq\})$	$\mid Trust_{i,j}$	
$S ::= Pro \mid Att$	$Att ::= Attest_i(\{Eq\})$	
$Pro ::= Proof(\{P\})$	$P ::= Att \mid Eq$	

[4] Other requirements would be relevant (such as the requirements concerning OBU) but we limit our example to the concerns of SP for the sake of conciseness.

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

The subscript j is introduced along with i to denote a component index. $Has_i(X)$ expresses the fact that variable X is an input variable located at component C_i (e.g. sensor or meter) and $Receive_{i,j}(\{S\},\{X\})$ specifies that component C_i can receive a set of declarations $\{S\}$ and variables $\{X\}$ from component C_j. These declarations can be proofs ($Proof(\{P\})$ with $\{P\}$ being equations or attestations) and attestations ($Attest_i(\{Eq\})$), that is to say simple declarations by a component C_i that properties $\{Eq\}$ are true. A component can also compute a variable defined by an equation $X = T$ (denoted by $Compute_i(X = T)$), check that a set of properties $\{Eq\}$ holds (denoted by $Check_i(\{Eq\})$), verify a proof 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 spot-check (i.e. request by a component C_j a value taken from array X and check that this value satisfies equations $\{Eq\}$, which is denoted by $Spotcheck_{i,j}(X,\{Eq\})$ with j a component index). Last but not least, trust assumptions are expressed using $Trust_{i,j}$ (meaning that component C_i trusts attestations from component C_j)[6].

The semantics $\mathcal{S}(A)$ of an architecture A is defined as the set of states of the components C_i of A resulting from compatible traces as described in [1]. A compatible trace contains only events that are instantiations of relations (such as $Has_i(X)$, $Receive_{i,j}(\{S\},\{X\})$, etc.) in A. The semantics $S(\phi)$ of a privacy property ϕ is defined as the set of architectures meeting ϕ. For example, $A \in S(Has^{none}(X))$ if and only if for all states $\sigma \in \mathcal{S}(A)$, the state σ_i is such that $\sigma_i(X) = \perp$, which expresses the fact that the component C_i cannot obtain the value of the variable X (or none of them in the case of an array). A sound and complete axiomatics has been defined to derive the integrity and privacy properties from the architecture. The decidability of this axiomatics depends on the reasoning power offered to the components.

An illustration of an architecture is given in Fig. 1. It relies on the relations defined previously to fulfill the functionality Ω. The OBU measures the locations *loc*, computes the global fee *fee*, and sends this latter to SP along with an attestation of integrity. SP trusts the declaration coming from OBU and verifies its authenticity. As can be seen, architectures provide an abstract, high-level view of a system. For example, we do not express at this level the particular method used by a component to verify that another component has actually attested (e.g. signed) a declaration. The objective at this stage is to be able to express and reason about the architecture rather than diving into technical details.

2.3 Design Strategies

In order to help designers finding their way among the variety of possible options, our approach is based on a succession of interaction steps. Each step consists in a question to the designer whose answer is used to trim the design space and drive the search for a suitable solution. The two key ingredients affecting the effectiveness of the process are the criteria to be used at each step and the order

[6] It can be noted this language is extensible with new relations to model other PETs as needed.

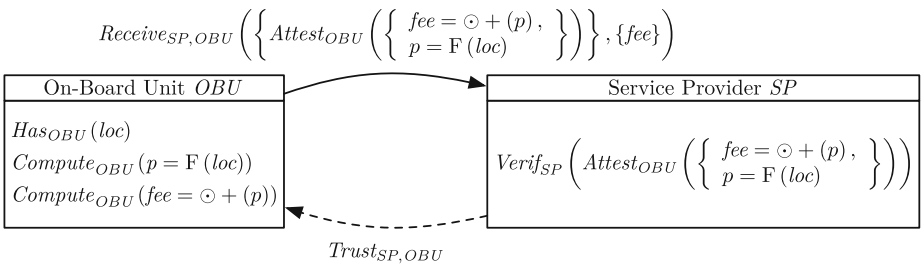

Fig. 1. Example of architecture relying on $Verif_i^{Attest}(Att)$ and $Trust_{i,j}$ primitives.

in which the questions should be asked. Based on our experience in the design of privacy preserving solutions, we propose the following strategies defined by steps 1 to 5 below. Steps 1 and 2, which have an overall effect on the possible solutions, are applied for the whole system. Then each equation of the definition of the functionality Ω is considered in turn in a bottom-up fashion[7] and Steps 3 to 5 are applied to each of them. Each choice adds a relation or a property to the architecture.

1. *Constraints:* The first question to be answered by the designer concerns the potential constraints imposed by the context or architectural choices that may have already been made by the designer. For example, the location of the input variables (e.g. sensors or meters) is often imposed by the environment and the absence of a direct communication channel between certain components may be a strong constraint. This criterion may typically have an impact on the occurrence of $Has_i\,(X)$, $Receive_{i,j}\,(\{S\}, \{X\})$, or $Compute_i\,(X = T)$ relations for metering, communications, or computations respectively.
2. *Anonymity:* The second question is the potential anonymity requirement. When anonymity is required, it can be expressed as a relationship between components. This type of requirement has an overall effect on the possible solutions because it introduces the need to implement anonymous channels between certain components and to specify the identifying values that have to be protected.
3. *Accuracy:* The first question for each equation is the level of accuracy required for the result. If an approximate solution is acceptable, then techniques such as perturbation, rounding (on individual values), aggregation, or sampling (on sets of values) can be considered (and the functionality Ω be augmented with a new function to perform this approximation). Otherwise these techniques are ruled out.
4. *Type of Trust:* The next question for each equation is the type of trust which can be accepted by the components, which is a key driver for the construction of a solution. We distinguish three types of trust:

[7] Starting from input variables is more efficient because their location is usually imposed by the context and they are often personal data.

(a) *Blind trust* is the strongest form of trust: if a component C_i blindly trusts a component C_j, it assumes that C_j will always behave as expected and all its attestations $Attest_j(\{Eq\})$ will be accepted as true after verification (by $Verif_i^{Attest}(Attest_j(\{Eq\}))$). This is expressed by a relation $Trust_{i,j}$ added to the architecture. Blind trust should obviously be used parsimoniously because it leads to the weakest solutions (technically speaking), or the solutions most vulnerable to misplaced trust. However, there are very reasonable (and even unavoidable) uses of blind trust, for example on the sensors providing the input values, or on secure hardware components. As far as techniques are concerned, this type of trust only requires authentication (e.g. for C_i to check that a message has indeed been sent by C_j: because C_j is assumed to be trustworthy, the content of his message will be accepted as such). Figure 1 depicts an architecture relying on blind trust.

(b) *Verifiable trust* (a posteriori verification) also considers by default that the trusted component behaves as expected but it is not as absolute as blind trust: it provides for the possibility of a posteriori verifications that this trust is not misplaced. Two types of techniques are typically used to implement this kind of trust: commitments (for the initial, and privacy preserving, declaration of the values) and spot-checks (for the verification of sample values). In this case, the architecture includes a $Spotcheck_{i,j}(X, \{Eq\})$ relation with X the array that can be sampled and $\{Eq\}$ the equations which should be satisfied (using the sampled X). The architecture illustrated in Fig. 2 relies on verifiable trust for the computation of $p = \mathrm{F}(loc)$.

(c) *Verified trust* (a priori verification) could be presented as a "non trust" option (or trust in the technology only) in the sense that a component C_i does not accept a statement as true if it is not able to verify it by itself (by a computation $Compute_i(X = T)$, a check $Check_i(\{Eq\})$ or the verification of a proof $Verif_i^{Proof}(Proof(\{P\}))$). Useful techniques to provide this level of guarantees include zero knowledge proofs, secure multi-party computations and homomorphic encryptions. The example in Fig. 2 relies on verified trust for the computation of $fee = \odot + (p)$ (through a homomorphic scheme relying on a hash function H).

5. *Assessment:* The last, but not least, question has to do with the preferences (e.g. in terms of performances, usability, or costs) that may lead to the rejection of certain solutions and with the detection of inconsistencies which may lead to the addition of new elements (e.g. a missing communication). For example, the limited computing power of a component, the low throughput of a communication channel, or the extra burden on the users can go against the use of certain $Receive_{i,j}(\{S\}, \{X\})$, $Compute_i(X = T)$, or $Verif_i^{Proof}(Pro)$ relations. This step is the counterpart of Step 1 (which concerns the a priori knowledge of the designer before the start of the design procedure): it leads to the filtering of the potential options resulting from the application of the previous steps of the procedure.

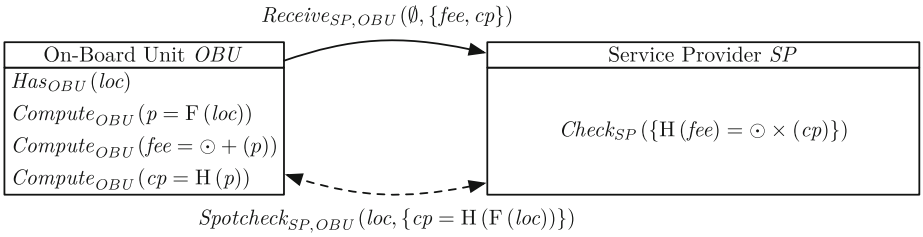

Fig. 2. Simplified *PrETP* electronic toll pricing architecture relying on verifiable trust for the computation of F and on verified trust for $\odot+$ (inspired by [2]).

These strategies guide the designer step-by-step through a succession of questions until an acceptable architecture is derived. Such architectures can be used in a purely informal way and represented as annotated graphs manipulated by designers who get an intuitive understanding of their meaning. However, this does not give strong guarantees that the obtained architectures really satisfy the privacy and integrity requirements of the system. One way to strengthen these guarantees is to rely on the formal framework detailed in [1].

This systematic design process is supported by a tool which is presented in the next section. This tool guides the designer and seamlessly builds a formal model of the architecture which can then be verified.

3 *CAPRIV* Tool

The *CAPRIV* computer aided privacy engineering tool has been developed to help non-expert designers to build architectures by following the strategies presented in this paper and to verify that the privacy requirements are met according to the formal model detailed in [1]. The interface and the back-end have been themselves developed with privacy by design in mind. For example, two components are not linked by any direct communication channel unless explicitly declared by the designer. The design of the tool makes it possible to hide the formal aspects of the model to the designer who does not want to be exposed to mathematical notations. This is mainly achieved through the use of a graphical user interface (GUI) and natural language statements. *CAPRIV* implements an iterative design procedure allowing the designer to come back to previous steps at any time. This section presents a functional description of the tool, followed by a brief overview of its implementation.

Functional Description. The GUI is divided into two parts: a Model and a View. The Model part is composed of three panes: Specify, Design, and Verify. The View part is composed of three other panes: Specification, Architecture, and Proofs. The View part shows the results of the interactions of the designer with the Model part.

Specification. The first task of the designer is to declare all the elements (components, variables, and functions) that will be used during the design process. Different properties of these elements, such as the fact that a variable is an array or a function is not invertible, can be selected throughout the process. The equations defining the functionality Ω are then declared (using these elements). Finally, the confidentiality and integrity requirements ϕ are defined based on the components and on the functionality. The current version of *CAPRIV* only supports simple equations (without function application nesting). This limitation can be circumvented by decomposing the functionality equations in simpler subequations.

Design. The next step for the designer is to build an architecture A meeting the requirements. To this aim, *CAPRIV* implements a design cycle following the strategies presented in Sect. 2.3. The designer must define the pre-existing constraints before choosing, for each equation in the functionality of the system, a location for the computation and a type of trust. Finally, the designer can add the missing communication links to make the architecture consistent. The Design panel is shown in Fig. 3.

Verification. If he chooses to do so, the designer can then verify whether the obtained architecture meets the requirements. The first verification concerns the consistency of the architecture (for example, the fact that the arguments

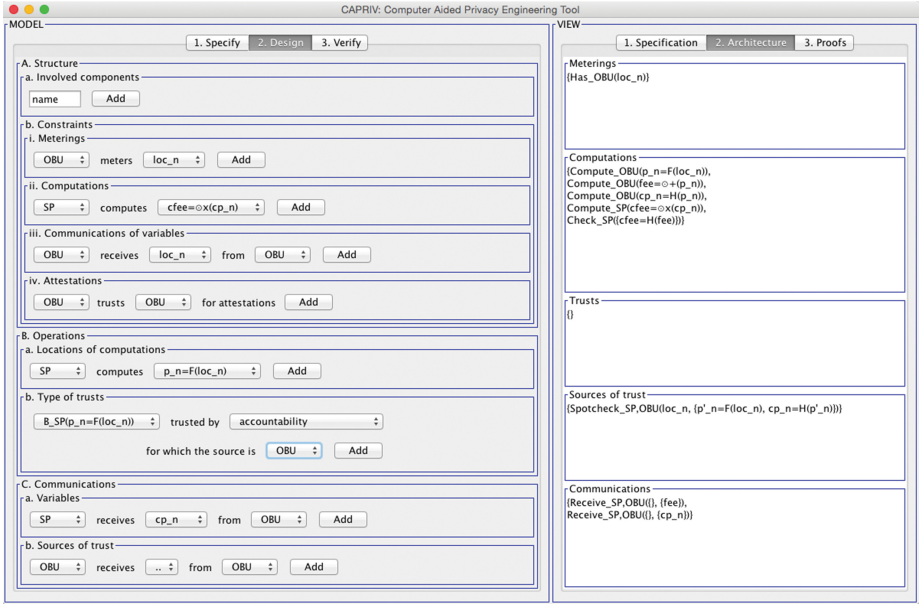

Fig. 3. Design view in *CAPRIV* at the end of the design process leading to the architecture depicted in Fig. 2.

of a computation must themselves be produced in one way or another and be available to the component performing the computation). The designer can then formally verify the satisfaction of the confidentiality and integrity requirements.

Implementation. Some of the verification features rely on external theorem provers and *CAPRIV* acts as a frontend interface for the *Why3* framework [4]. *Why3* is a platform in which theories can be expressed in an *ML*-flavored language. Standard libraries are provided to easily model structures such as rings and arrays for instance. *Why3* relies itself on external provers such as *Alt-Ergo* [5] or *Z3* [23] to prove theorems. *CAPRIV* automatically generates theories corresponding to the architecture (as axioms and conjectures to be proven), then calls *Why3*, and finally handles its answer.

When the expected property cannot be proved, an advanced designer can choose to use the *Why3* GUI offering an interactive theorem proving environment in which specific provers can be called with the possibility to apply different solving strategies. An expert designer can even exploit the detailed configuration or modify the theories — but these skills are not expected for a standard use of *CAPRIV*. This choice to rely on external tools for some parts of the verification shows that it is possible to integrate privacy by design methodologies with existing formal verification tools.

4 Electronic Toll Pricing Case Study

In this section, we apply the methodology presented in Subsect. 2.3 to the electronic toll pricing example introduced in the previous sections.

Architecture Requirements. As a reminder, the functionality for our case study is expressed by $\Omega = \{fee = \odot + (p), p = \mathrm{F}(loc)\}$. The privacy requirements are $Has^{one}_{SP}(loc)$, $Has^{one}_{SP}(p)$, and $Has^{all}_{SP}(fee)$ for the confidentiality properties and $K_{SP}(fee = \odot + (p))$ and $B_{SP}(p = \mathrm{F}(loc))$ for the integrity properties as defined in Subsect. 2.1.

Architecture Design

1. Constraints. The first task of the designer is to identify the unavoidable constraints that must be taken into account in the design of the system. For this case study, the locations are measured by the on-board units: $Has_{OBU}(loc)$. The designer has also to make explicit other predefined choices (either imposed by the customer or resulting from his own knowledge or experience). We assume here that he chooses to locate the computations on the on-board units to minimise personal data disclosure as follows: $Compute_{OBU}(p = \mathrm{F}(loc))$ and $Compute_{OBU}(fee = \odot + (p))$. Another obvious constraint is for the provider to get the fee: $Receive_{SP,OBU}(\emptyset, \{fee\})$.

2/3. Anonymity and Accuracy. No anonymity channel is required by the designer for this architecture and no approximation technique can be applied because the fee has to be computed accurately.

4. Type of Trust. The key step in the process is the choice of the types of trust accepted by the parties for each part of the functionality (the computation of individual prices and their sum). We assume that verifiable trust is acceptable for the service provider (which is consistent with the use of the privacy requirement B_{SP} defined previously). Pricing will therefore be checked a posteriori by comparing a sample of the actual locations *loc* with the corresponding commitments *cp* (using a one-way homomorphic hash function H) sent by the driver (in fact his *OBU*): $Spotcheck_{SP,OBU}(loc, \{cp = \mathrm{H}(\mathrm{F}(loc))\})$ and $Receive_{SP,OBU}(\emptyset, \{cp\})$. The homomorphism property of the function H (such that $\mathrm{H}(\Sigma_{0\leq i\leq n}(T_i)) = \Pi_{0\leq i\leq n}(\mathrm{H}(T_i))$ for all terms T_i) enables the provider to check the integrity property for the computation of $fee = \odot + (p)$ by verifying that the product of the committed prices is equal to the hashed fee as follows: $Check_{SP}(\{\mathrm{H}(fee) = \odot \times (cp)\})$.

5. Assessment. The last tasks for the designer are to check the consistency of the architecture and, if necessary, to add the missing elements to get a consistent architecture. In our example, it is necessary to add $Compute_{OBU}(cp = \mathrm{H}(p))$ to ensure that a component is in charge of computing the *cp* variables.

Figure 3 illustrates the design view of *CAPRIV* when the choices made previously have been input. Figure 2 pictures the architecture (which is a simplified version of [2]) obtained at the end of the design process. This latter is expressed in the formal language detailed in [1].

Architecture Verification. The designer can then formally verify that the architecture meets the requirements. This verification can be made (either automatically or interactively) using the *CAPRIV* tool sketched in Sect. 3 which implements the axiomatics presented in [1].

The solution designed here relies on heavy on-board units able to perform the billing computations. Moreover, it assumes a direct link between the on-board unit and the provider: the driver has to trust the on-board unit not to disclose too much data to the provider. This issue could be solved by adding a proxy under the control of the driver which would filter the communications between *SP* and *OBU*. This alternative can be expressed in the same framework by adding another component but space considerations prevent us from presenting it here.

5 Related Work

Several authors [12,17,19,24,30] have already pointed out the complexity of "privacy engineering" as well as the "richness of the data space" [12] calling for the development of more general and systematic methodologies for privacy by design. As far as privacy mechanisms are concerned, [17,21] 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, [17] proposes a number of guidelines for the design of compilers for secure computation and zero-knowledge proofs whereas [9] 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), [22] also proposes implementation techniques to make it easier for developers to take into account privacy and security requirements. Finally, [31] proposes a development method for security protocols allowing to derive a protocol by refinement. However, this method does not offer decision support for the designer to choose among different possibilities as we do.

The recent proposal [18] also emphasizes the importance of architectures for privacy by design. Reference [18] proposes a design methodology for privacy (inspired by [3]) based on tactics for privacy quality attributes (such as minimisation, enforcement or accountability) and privacy patterns (such as data confinement, isolation or Hippocratic management). The work described in [18] is complementary to the approach presented here: [18] does not consider formal aspects while this paper does not address the tactics for privacy by design.

Design patterns are used in [14] to define eight privacy strategies[8] called respectively: Minimise, Hide, Separate, Aggregate, Inform, Control, Enforce and Demonstrate. Other authors put forward pattern-based approaches: [13] proposes a language for privacy patterns allowing for a designer to choose relevant PETs; [29] describes a solution for online interactions; at a higher level, [25] 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.

All the aforementioned work is very helpful and paves the way for a wider adoption of privacy by design. We believe however that there is still a gap between techniques or methods (such as design patterns or tactics) which are described informally at a very high abstraction level and formal models of privacy that usually address precise technical issues or specific requirements (such as protocols dedicated to smart metering, electronic toll pricing, or electric vehicle charging). The former are intended as guidelines for software designers and engineers but do not provide any formal guarantees; the latter provide formal guarantees but they are very specific and can hardly be used by software engineers to build a new product. Moreover, they are generic frameworks and they do not include any specific privacy by design methodology.

Filling this gap is precisely the objective of this paper. Previous work on this very topic is scarce. One exception is the framework introduced in [19] which defines the meaning of the available operations in a (trace-based) operational semantics and proposes an inference system to derive properties from architectures. Even though the goal of [19] is to deal with architectures, it remains at a lower level of abstraction than the framework sketched here and it can hardly

[8] Strategies are defined as follows in [14]: "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".

be extended to other privacy mechanisms. In addition, it is not associated with design strategies as proposed in Sect. 2 of this paper. Complementary work by the authors are [1] which presents the formal framework (language of architectures and requirements, semantics, and axiomatics) and illustrates it with a smart metering example, and [32] which completes the formal framework with a link between architectures and actual implementations (as protocols). Any protocol consistent with an architecture would then meet the properties of the architecture as defined in this paper.

6 Conclusion

Considering that there is usually no absolute notion of personal data minimality, the only solution is to specify the requirements of the parties and try to find a solution to meet them all or to iterate otherwise. For example, in the electronic toll pricing case study discussed in Sect. 4, a solution has been found in which the only personal data disclosed to the provider is the fee to be paid by the driver (which can harldy be avoided) and occasionally (when a spot-check is initiated) the position of the vehicle. Other solutions can be found which do not involve spot-checks but rely on more expensive secure on-board units.

In addition to its interest in the design phase, the use of the methodology proposed here provides a key benefit in terms of accountability which will become an obligation with the new GDPR [8]. Accountability is defined in the Article 22 as the following obligation for data collectors: "The controller shall adopt appropriate policies and implement appropriate and demonstrable technical and organisational 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...". A significant byproduct of the approach described in this paper is to provide to data collectors a documented and rigorous justification of the design choices, which will become a key asset for the implementation of their accountability requirements.

Another benefit of the approach presented here is that designers do not have to opt from the outset for a formal framework. Rather, they can first explore the design space based on initial inputs provided in a non formal language and analyse the suggested architectures based on their graphical representations. They can content themselves with this step or wish to go beyond and try to prove other properties of their architectures. In the latter case, depending on their level and type of expertise, they can either rely on an automatic verification mode or choose among the verification tools integrated within the design environment.

For the reasons discussed in Sect. 2, the approach described in this paper focuses on architectures. An extension of this work is the integration of other types of trust such as the trust in pairs, in particular trust conditional on the endorsement of a declaration by a minimal number (or ratio) of pairs. From an academic perspective, another avenue for further research is the use of the formal framework presented here to provide a classification of solutions presented in the literature (in the style of [15]) based on formal criteria.

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. Antignac, T., Le Métayer, D.: Privacy architectures: reasoning about data minimisation and integrity. In: Mauw, S., Jensen, C.D. (eds.) STM 2014. LNCS, vol. 8743, pp. 17–32. Springer, Heidelberg (2014)
2. Balasch, J., Rial, A., Troncoso, C., Geuens, C.: PrETP: privacy-preserving electronic toll pricing. In: Proceedings of the 19th USENIX Security Symposium, Washington, DC, August 2010, pp. 63–78 (2010)
3. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. SEI series in Software Engineering, 3rd edn. Addison-Wesley, Reading (2012)
4. Bobot, F., Filliâtre, J.c., Marché, C., Paskevich, A.: Why3: shepherd your herd of provers. In: Workshop on Intermediate Veri Cation Languages (2011)
5. Conchon, S., Contejean, E.: The Alt-Ergo automatic theorem prover (2008). http://alt-ergo.lri.fr/
6. Deswarte, Y., Aguilar Melchor, C.: Current and future privacy enhancing technologies for the internet. Ann. Des Télécommun. **61**(3–4), 399–417 (2006)
7. Diaz, C., Kosta, E., Dekeyser, H., Kohlweiss, M., Girma, N.: Privacy preserving electronic petitions. Identity Inf. Soc. **1**(1), 203–209 (2009)
8. European Parliament: general data protection regulation, ordinary legislative procedure: first reading, March 2014
9. Fournet, C., Kohlweiss, M., Danezis, G., Luo, Z.: Zql: a compiler for privacy-preserving data processing. In: Proceedings of the 22Nd USENIX Conference on Security, SEC 2013, pp. 163–178. USENIX Association, Berkeley (2013)
10. Garcia, F.D., Jacobs, B.: Privacy-friendly energy-metering via homomorphic encryption. In: Cuellar, J., Lopez, J., Barthe, G., Pretschner, A. (eds.) STM 2010. LNCS, vol. 6710, pp. 226–238. Springer, Heidelberg (2011)
11. Goldberg, I.: Privacy enhancing technologies for the internet III: ten years later. In: Digital Privacy: Theory, Technologies, and Practices, pp. 3–18. Auerbach Publications (2007)
12. Gürses, S., Troncoso, C., Diaz, C.: Engineering privacy by design. In: Presented at the Computers, Privacy and Data Protection Conference (2011)
13. Hafiz, M.: Pattern language for developing privacy enhancing technologies. Softw. Pract. Exp. **43**(7), 769–787 (2010)
14. Hoepman, J.H.: Privacy design strategies. In: Cuppens-Boulahia, N., Cuppens, F., Jajodia, S., Abou El Kalam, A., Sans, T. (eds.) SEC 2014. IFIP AICT, vol. 428, pp. 446–459. Springer, Heidelberg (2014)
15. Jawurek, M., Kerschbaum, F., Danezis, G.: Privacy technologies for smart grids - a survey of options. Technical report MSR-TR-2012-119, Microsoft, November 2012
16. de Jonge, W., Jacobs, B.: Privacy-friendly electronic traffic pricing via commits. In: Degano, P., Guttman, J., Martinelli, F. (eds.) FAST 2008. LNCS, vol. 5491, pp. 143–161. Springer, Heidelberg (2009)
17. Kerschbaum, F.: Privacy-preserving computation (position paper). In: Presented at the Annual Privacy Forum Conference (2012)
18. Kung, A.: PEARs: privacy enhancing architectures. In: Preneel, B., Ikonomou, D. (eds.) APF 2014. LNCS, vol. 8450, pp. 18–29. Springer, Heidelberg (2014)

19. Le Métayer, D.: Privacy by design: a formal framework for the analysis of architectural choices. In: Proceedings of the Third ACM Conference on Data and Application Security and Privacy, CODASPY 2013, pp. 95–104. ACM, New York (2013)
20. LeMay, M., Gross, G., Gunter, C.A., Garg, S.: Unified architecture for large-scale attested metering. In: 40th annual Hawaii International Conference on System Sciences (HICSS 2007), pp. 115–124, January 2007
21. Maffei, M., Pecina, K., Reinert, M.: Security and privacy by declarative design. In: 2013 IEEE 26th Computer Security Foundations Symposium (CSF), pp. 81–96 (2013)
22. Manousakis, V., Kalloniatis, C., Kavakli, E., Gritzalis, S.: Privacy in the cloud: bridging the gap between design and implementation. In: Franch, X., Soffer, P. (eds.) CAiSE Workshops 2013. LNBIP, vol. 148, pp. 455–465. Springer, Heidelberg (2013)
23. de Moura, L., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008)
24. Mulligan, D.K., King, J.: Bridging the gap between privacy and design. Univ. Pa. J. Const. Law **14**(4), 989–1034 (2012)
25. Pearson, S., Benameur, A.: A decision support system for design for privacy. In: Fischer-Hübner, S., Duquenoy, P., Hansen, M., Leenes, R., Zhang, G. (eds.) Privacy and Identity Management for Life. IFIP AICT, vol. 352, pp. 283–296. Springer, Heidelberg (2011)
26. Popa, R.A., Balakrishnan, H., Blumberg, A.J.: VPriv: protecting privacy in location-based vehicular services. In: Proceedings of the 18th USENIX Security Symposium, Montreal, Canada, August 2009, pp. 335–350 (2009)
27. Rezgui, A., Bouguettaya, A., Eltoweissy, M.Y.: Privacy on the web: facts, challenges, and solutions. IEEE Secur. Priv. **1**(6), 40–49 (2003)
28. Rial, A., Danezis, G.: Privacy-Preserving smart metering. Technical report MSR-TR-2010-150, Microsoft Research, November 2010
29. Romanosky, S., Acquisti, A., Hong, J., Cranor, L.F., Friedman, B.: Privacy patterns for online interactions. In: Proceedings of the 2006 Conference on Pattern Languages of Programs, PLoP 2006. pp. 12:1–12:9. ACM, New York (2006)
30. Spiekermann, S., Cranor, L.F.: Engineering privacy. IEEE Trans. Softw. Eng. **35**(1), 67–82 (2009)
31. Sprenger, C., Basin, D.: Developing security protocols by refinement. In: Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, pp. 361–374. ACM, New York (2010)
32. Ta, V.T., Antignac, T.: Privacy by design: on the conformance between protocols and architecture. In: Cuppens, F., Garcia-Alfaro, J., Zincir Heywood, N., Fong, P.W.L. (eds.) FPS 2014. LNCS, vol. 8930. Springer, Heidelberg (2015)
33. Troncoso, C., Danezis, G., Kosta, E., Preneel, B.: Pripayd: privacy friendly pay-as-you-drive insurance. In: Ning, P., Yu, T. (eds.) Proceedings of the 2007 ACM Workshop on Privacy in the Electronic Society, WPES 2007, pp. 99–107. ACM, New York (2007)
34. Tschantz, M.C., Wing, J.M.: Formal methods for privacy. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009. LNCS, vol. 5850, pp. 1–15. Springer, Heidelberg (2009)