Trust Driven Strategies for Privacy by Design (Long Version)

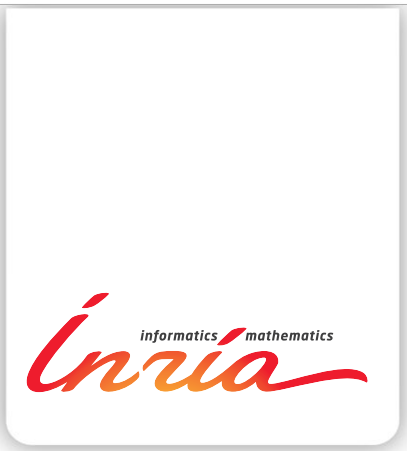
Thibaud Antignac, Daniel Le Métayer

Trust Driven Strategies for Privacy by Design (Long Version)

Thibaud Antignac, Daniel Le Métayer

RESEARCH REPORT
N° 8676
February 2015
Project-Team Privatics

ISSN 0249-6399 ISRN INRIA/RR--8676--FR+ENG



Trust Driven Strategies for Privacy by Design (Long Version)

Thibaud Antignac, Daniel Le Métayer

Project-Team Privatics

Research Report n° 8676 — February 2015 — 18 pages

Abstract: In this report, we describe a multi-stage approach for privacy by design. The main design step is the choice of the types of trust that can be accepted between the stakeholders. Architectures can be initially defined in a purely informal way and then mapped into a formal dedicated model.

Key-words: trust, privacy, design, logics, software engineering, architectures

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

Stratégies de Respect de la Vie Privée par Construction Guidées par la Confiance (Version Longue)

Résumé : Dans ce rapport, nous décrivons une approche multi-étapes pour le respect de la vie privée par construction. La principale étape de conception est le choix entre différents types de confiance qui peuvent être acceptés par les parties. Les architectures sont initialement définies de manière purement informelle et peuvent être reliées à un modèle formel dédié.

Mots-clés : confiance, respect de la vie privée, logiques, génie logiciel, architectures

1 Introduction

The integration of formal specifications within more general semi-formal or informal frameworks is a nagging problem and a key factor for the adoption of more rigorous development methods in software engineering. In this report, we describe a multi-stage approach for such an integration applied to privacy by design. 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 even become a legal obligation in the European Community if the current draft of the Data Protection Regulation [10] 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 it 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, witch 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 techniques have been proposed during the last decades (including zero-knowledge proofs, secure multi-party computation, homomorphic encryption, etc.) Each of these techniques provides different guarantees based on different assumptions and therefore are 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.

In practice, privacy by design is often a matter of choice [23]: multiple options are generally available to achieve a given set of requirements, 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. On the basis of the above, we believe that the most urgent needs in this area are:

1. The availability of design environments providing strategies for the search of solutions based on the different requirements of the system and the available privacy enhancing tools.
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 a 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 report, 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 Section 2 and sketch the formal framework in Section 3. In Section 4, we apply the approach to a case study, an electronic toll pricing system. Section 5 discusses related work and Section 6 outlines directions for further research. In Appendix A, we outline the implementation and use of CAPRIV, our computer assisted privacy engineering tool.

2 Trust Driven Strategies

A wide range of privacy enhancing technologies (PETs) are now available, which can provide strong privacy guarantees in a variety of contexts [8, 14, 21, 32]. 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 [9, 38]. 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 (Subsection 2.1) before defining our notion of architecture (Subsection 2.2) and describing the overall strategy and criteria used for the construction of a privacy compliant architecture (Subsection 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 payment systems [37, 20, 30] have to guarantee both the correctness of the computation of the fee and the limitation of the collection of location data; smart metering systems [13, 25, 33] 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.

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

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

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 report 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 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 is expressed as the computation of a set of equations Ω[3] such that $\Omega = \left\{\tilde{X} = T\right\}$ with terms T defined as follows:

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

where X represents variables ($X \in \mathit{Var}$), K index variables ($K \in \mathit{Index}$), Cx constants ($Cx \in \mathit{Const}$), Ck index constants ($Ck \in \mathit{Nat}$[4]), F functions ($F \in \mathit{Fun}$) and $\odot F(X)$ is the iterative application of function F to the elements of the array denoted by X (e.g. sum of the elements of X if f is equal to $+$). We assume that each array variable X represents an array of fixed size $\mathit{Range}(X)$.

2.2 Architectures

Many definitions of architectures have been proposed in the literature. In this report, we adopt a definition inspired by [6][5]: *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.* Following this principle, a set of components C_i, $i \in [1, \ldots, n]$ is associated with relations describing their capabilities. These capabilities depend on the set of available PETs. For the purpose of this report, we consider the architecture language described in Table 1 [3].

Subscripts i and j are component indexes and the notation $\{Z\}$ is used to define a set of terms of category Z. $\mathit{Has}_i(X)$ expresses the fact that variable X is an input variable located at component C_i (e.g. sensor or meter) and $\mathit{Receive}_{i,j}(\{S\}, \{X\})$ specifies that component C_i can receive from component C_j messages consisting of a set of statements $\{S\}$ and a set of variables $\{X\}$. A statement can be either a proof of a property P (denoted by $\mathit{Proof}_i(P)$) or an attestation (denoted by $\mathit{Attest}_i(\{Eq\})$), that is to say a simple declaration 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 $\mathit{Compute}_i(X = T)$, check that a set of properties Eq holds (denoted by $\mathit{Check}_i(\{Eq\})$, verify a proof of a property Pro received from another component (denoted by $\mathit{Verif}_i^{Proof}(\mathit{Pro})$) or the origin of an attestation (denoted by $\mathit{Verif}_i^{Attest}(\mathit{Att})$), or perform a spotcheck (i.e. request from a component C_j a value X_k taken from array X and check that this value satisfies property Eq , which is denoted by $\mathit{Spotcheck}_{i,j}(X_k, Eq)$). Primitive properties Eq are simple equations on terms T. Last but not least, trust assumptions are expressed using $\mathit{Trust}_{i,j}$ (meaning that component i trusts component j).

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

[4] Set of natural numbers.

[5] This definition is a generalisation (to system architectures) of the definition of software architectures proposed in [6].

Table 1: Privacy Architecture Language.

$$
\begin{array}{lll}
A ::= \{R\} & & \\
R ::= Has_i\left(\tilde{X}\right) & \mid Receive_{i,j}\left(\{S\},\{\tilde{X}\}\right) & \\
\quad \mid Compute_i\left(\tilde{X}=T\right) & \mid Check_i\left(\{Eq\}\right) & \\
\quad \mid Verif_i^{Proof}\left(Pro\right) & \mid Verif_i^{Attest}\left(Att\right) & \\
\quad \mid Spotcheck_{i,j}\left(X_k, Eq\right) & \mid Trust_{i,j} & \\
 & & \\
S ::= Pro \mid Att & & Att ::= Attest_i\left(\{Eq\}\right) \\
Pro ::= Proof_i\left(\{P\}\right) & & Eq ::= T_1\ Rel\ T_2 \\
P ::= Att \mid Eq & & Rel ::= = \mid < \mid > \mid \leq \mid \geq
\end{array}
$$

As an illustration, Figure 1 pictures a simple architecture involving a toll service provider *TSP* and a driver represented by an on-board unit *OBU*. The *OBU* measures the trajectory parts TP_n, computes the fee *fee* and commitments cp_n on the individual costs p_n. Both the fee and the commitments are sent to the TSP which can initiate spot checks on the individual costs. The design process leading to this architecture and its motivation are detailed in Section 4.

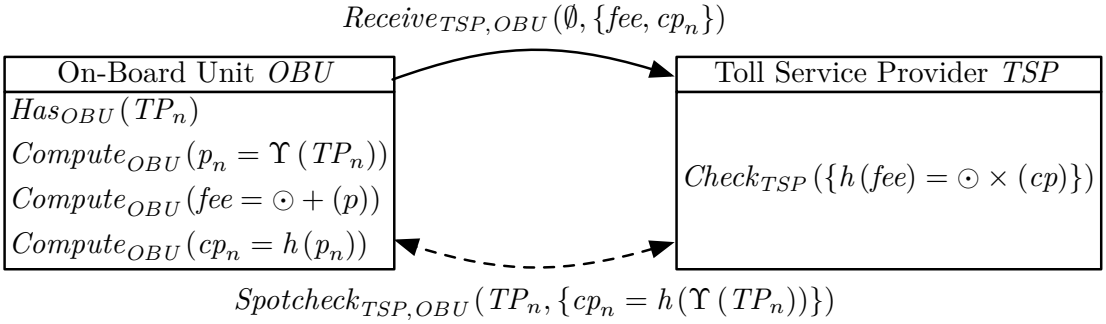


Figure 1: Simplified PrETP electronic toll pricing architecture inspired by [5].

As shown in Figure 1, 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 achieve a proof or to verify it, or to check that another component has actually attested (e.g. signed) a property. Another main departure from protocol specification languages is that we do not have any specific ordering or notion of sequentiality here, even though functional dependencies introduce implicit constraints in the events of the system. The objective at this stage is to be able to express and reason about the main design choices rather than diving into technical details.

2.3 Design Strategy

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 in which the questions should be asked. Based on our experience in the design of privacy preserving solutions, we propose the following strategy defined by steps 1 to 5 below. Step 1 and 2, which have an overall effect on the possible solutions, are applied for the whole

system. Then each equation $X = \mathrm{F}(T_1, \ldots, T_n)$ of the definition of the service is considered in turn in a bottom-up fashion[6] and Steps 3 to 5 are applied to each of them.

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 direct communication channel between certain components may be a strong constraint. This criterion may typically have an impact on the occurrence of Has_i, $Receive_{i,j}$ or $Compute_i$ relations.

2. *Anonymity:* The second question is the potential anonymity requirement. When anonymity is required, it can be expressed as a relationship between actors (components here). 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. 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:

   (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 statements will be accepted as true. This is expressed by the relation $Trust_{i,j}$ in our language of architectures. 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 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).

   (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). An example of a partial architecture relying on verifiable trust is proposed in Figure 1.

   (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 does not accept a statement as true if it is not able to verify it by itself. Useful techniques to provide this level of guarantees include zero knowledge proofs, secure multi-party computation and homomorphic encryption. In this report, we will consider essentially zero knowledge proofs (through the $Receive_{i,j}(Proof_k(P), \{X\})$ and $Verif_i^{Proof}(Proof_j(P))$ relations).

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

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 $Compute_i$, $Receive_{i,j}$, or $Proof_i$ 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.

3 Formal Framework

The strategy presented in the previous section guides the designer step-by-step through a succession of questions until one (or several) acceptable architectures are derived. Architectures are described so far in a purely informal way, as sets of relations. They can be represented as annotated graphs and manipulated by designers who can get an intuitive understanding of their meaning. However, there is no strong guarantee at this stage that the obtained architectures really satisfy the privacy and integrity requirements of the system. One way to strengthen these guarantees is to provide a formal framework to define architectures and to reason about their properties. In this section, we sketch the formal framework introduced in [3], including the privacy logic and an overview of its axiomatics.

Because privacy is closely connected with the notion of knowledge, epistemic logics [11] form an ideal basis to reason about privacy properties. However standard epistemic logics based on possible worlds semantics suffer from a weakness which makes them unsuitable in the context of privacy: this problem is often referred to as "logical omniscience" [17]. It stems from the fact that agents know all the logical consequences of their knowledge (because these consequences hold in all possible worlds). An undesirable outcome of logical omniscience would be that, for example, an agent knowing the hash $H(v)$ of a value v would also know v. This is obviously not the intent in a formal model of privacy where 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 at hand (e.g. knowledge, zero-knowledge proof, trust, etc.). In this report, we take inspiration from the "deductive algorithmic knowledge" approach [11, 31] in which the explicit knowledge of a component C_i is defined as the knowledge that it can actually compute using his own deductive system $\triangleright_i$. Another relation, Dep_i, is introduced to express that a variable can be derived from other variables. $Dep_i\left(\tilde{X}, \left\{\tilde{X}^1, \ldots \tilde{X}^n\right\}\right)$ means that a value for $\tilde{X}$ can be obtained by C_i $(\exists F, \tilde{X} = F(\tilde{X}^1, \ldots, \tilde{X}^n))$. The absence of a relation such as $Dep_i(x_k, \{y_k\})$ prevents component C_i from deriving the value of x_k from the value of y_k, capturing the hiding property of the hash application $y_k = H(x_k)$. The syntax of the logic is defined in Table 2.

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

Table 2: Architecture logic.

This logic involves two modalities, denoted by K_i and B_i, which represent respectively knowledge and belief properties of a component C_i. The logic can be used to express useful properties of architectures: for example $Has_i^{all}(X)$ expresses the fact that component C_i can obtain or derive (using its deductive system $\triangleright_i$) the value of X_k for all k in $Range(X)$. $Has_i^{one}(X)$ expresses the fact that component C_i can obtain or derive the value of X_k for at most one k in $Range(X)$. Finally, $Has_i^{none}(X)$ is the privacy property stating that C_i does not know any X_k value. Integrity properties can be expressed as $K_i(Eq)$ properties, meaning that component C_i can know (establish with certainty) that equation Eq holds. It should be noted that Has_i 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. Integrity requirements can be expressed using the $K_i(Eq)$ and $B_i(Eq)$ properties. $K_i(Eq)$ means that component C_i can establish the truthfulness of Eq while $B_i(Eq)$ expresses the fact that C_i can test this truthfulness, and therefore detect its falsehood (or believe that the property is true otherwise).

The semantics $S(\Phi)$ of a property Φ is defined in [3] as the set of architectures meeting Φ. This definition relies in turn on the semantics of architectures which is defined as the set of states of the components of the system produced by all the traces (sequences of events) that are compatible with the architecture. For example, $A \in S(Has_i^{none}(X)) \Leftrightarrow \forall \sigma \in \mathcal{S}(A), \sigma_i^v(X) = \perp$, which means that an architecture A meets property $Has_i^{none}(X)$ if and only if X is undefined in all possible states of a system complying with A. An axiomatics has been defined for this language of properties and both its correctness and its completeness have been established. In addition, if $\triangleright_i$ is decidable, then so is the axiomatics, which can therefore be used as a basis for the Computer Aided Privacy Engineering system described in Appendix A.

4 Electronic Toll Pricing Case Study

In this section, we apply the methodology presented in Subsection 2.3 to an electronic toll pricing application. Electronic toll pricing allows drivers to be charged depending on their actual behavior (mileage, time, ...). The fee (*fee*) due at the end of the billing period is based on their use of the road infrastructures modeled as trajectory parts (TP_n for a period of time n). The service $fee = \sum_n (\Upsilon(TP_n))$ (where Υ stands for the pricing function) is expressed in our language as $\Omega = \{fee = \odot + (p), p_n = \Upsilon(TP_n)\}$. We assume here that the architecture A should allow the toll service provider (TSP) to receive and to verify a posteriori the integrity of the fees declared by the drivers through the on-board unit (OBU) installed in their car.

Architecture Requirements.

The provider must get access to the global fee: $A \vdash Has_{TSP}^{all}(fee)$. Moreover, we assume that drivers do not want the provider to get access to the details of their trajectories (TP_n) or intermediate prices (p_n). However, the access by the provider to a sample of these values is allowed if needed for a posteriori integrity verification: $A \vdash Has_{TSP}^{one}(TP) \wedge Has_{TSP}^{one}(p)$. Moreover, the architecture must ensure that the provider can believe that the value sent by the driver for *fee* is correct: $A \vdash B_{TSP}(fee = \odot + (\Upsilon(TP_n)))$.

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 trajectory parts are measured by the on-board units: $Has_{OBU}(TP_n)$. 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: $Compute_{OBU}(p_n = \Upsilon(TP_n))$ and $Compute_{OBU}(fee = \odot + (p))$. Another obvious constraint is for the provider to get the fee: $Receive_{TSP,OBU}(\emptyset, \{fee\})$.

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 computed accurately.

4. Type of Trust. The key step in the strategy is the choice of the types of trust accepted by the parties for each part of the service (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 believe modality B_{TSP} in the requirements defined previously. Pricing will therefore be checked a posteriori by comparing a sample of the actual trajectory parts TP_n with the corresponding commitment (using a one-way hash function h) sent by the driver (in fact his OBU): $Spotcheck_{TSP,OBU}(TP_n, \{cp_n = h(\Upsilon(TP_n))\})$ and $Receive_{TSP,OBU}(\emptyset, \{cp_n\})$. The homomorphism property ($\{h(x) = \odot \times (h(y))\} \triangleright x = \odot + (y)$) enables the provider to check the integrity property by verifying that the product of the committed prices is equal to the hashed fee: $Check_{TSP}(\{h(fee) = \odot \times (cp)\})$.

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 $Compute_{OBU}(cp_n = h(p_n))$ to ensure that a component is in charge of computing the cp_n variables.

As discussed in Section 2, the dependence relations have to be defined to model the computational power of the components *OBU* and *TSP*. We assume that they both have the same dependence relation here for the sake of simplicity, noted Dep_i for $i \in \{OBU, TSP\}$. The relations are such that $(fee, \{p\}) \in Dep_i$, $(p_n, \{TP_n\}) \in Dep_i$, $(TP_n, \{p_n\}) \in Dep_i$, and $(cp_n, \{p_n\}) \in Dep_i$. The reason is that we assume that only the sum and the hash function are not easily invertible here, so we have $(p_n, \{fee\}) \notin Dep_i$ and $(p_n, \{cp_n\}) \notin Dep_i$). The deduction capabilities $\triangleright_i$ model the homomorphism property and the standard equality properties over terms ($\emptyset \triangleright_i t = t$, $\{t = u\} \triangleright_i u = t$, $\{t = u, u = v\} \triangleright_i t = v$, and $\{t = u, Eq\} \triangleright_i Eq[t/u]$ with u free in Eq).

The architecture obtained at the end of the above design process (which is a simplified version of [5]) is pictured in Figure 1.

Architecture Verification.

If he chooses to do so, the designer can then formally verify that the architecture resulting from the application of the above design steps meets its requirements. This verification can be made (either automatically or interactively) using the CAPRIV tool sketched in Appendix A which implements the axiomatics presented in [3]. We just provide here the intuition of the proof for this case study. The spotcheck over TP_n makes it possible to derive $B_{TSP}(cp_n = h(\Upsilon(TP_n)))$, hence to prove that the provider can believe that the pricing function as been correctly applied. Checking that $h(fee) = \odot \times (cp)$ allows him to know (and also to believe because knowledge is stronger than belief) that the equation holds. By substitution, we then have $B_{TSP}(h(fee) = \odot \times (h(\Upsilon(TP))))$ and therefore $B_{TSP}(fee = \odot + (p))$ thanks to the homomorphism of h, which proves that the provider can believe that the fee provided by the driver is correct.

Using the dependence relation (which expresses the one-way behavior of the aggregation and of the hash function), it can be shown that the communication to the provider of the fee and the commitments on the prices does not lead to the disclosure of any other personal data. Only a limited amount of the trajectory parts is disclosed to enable the implementation of verifiable trust, which leads to $A \vdash Has^{one}_{TSP}(TP)$ as required.

The solution chosen here for the sake of conciseness 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 an additional component acting as a proxy under the control of the driver which would filter the communications between the provider and the meter. This alternative can be expressed in the same framework but space considerations prevent us from presenting it here.

5 Related Work

Several authors [15, 21, 24, 28, 35] have already pointed out the complexity of "privacy engineering" as well as the "richness of the data space"[15] calling for the development of more general and systematic methodologies for privacy by design. As far as privacy mechanisms are concerned, [21, 26] point 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, [21] proposes a number of guidelines for the design of compilers for secure computation and zero-knowledge proofs whereas [12] 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), [27] also proposes implementation techniques to make it easier for developers to take into account privacy and security requirements. Finally, [36] 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.

A recent proposal ([22]) also emphasizes the importance of architectures for privacy by design. [22] proposes a design methodology for privacy (inspired by [6]) 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 [22] is complementary to the approach presented here: [22] does not consider formal aspects while this report does not address the tactics for privacy by design.

Design patterns are used in [18] to define eight privacy strategies[7] called respectively: Minimise, Hide, Separate, Aggregate, Inform, Control, Enforce and Demonstrate. Other authors put forward pattern-based approaches: [16] proposes a language for privacy patterns allowing for a designer to choose relevant PETs; [34] describes a solution for online interactions; at a higher level, [29] 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

[7]Strategies are defined as follows in [18]: "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."

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. General frameworks already exist to ease the use of formal methods such as the B method [2], Z [1], and VDM [7] for instance. However, they are generic frameworks and they do not include any specific privacy by design methodology. Filling this gap is precisely the objective of this report. Previous work on this very topic is scarce. One exception is the framework introduced in [24] 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. Even though the goal of [24] is to deal with architectures, it remains at a lower level of abstraction than the framework sketched here and it can hardly be extended to other privacy mechanisms. In addition, it is not associated with a design strategy as proposed in Section 2 of this report. Other related work by the authors are [4], which is a position paper advocating the role of architectures in privacy by design, and [3] which presents the formal framework (language of architecture, semantics and axiomatics) and illustrates it with a smart metering example. The main contribution of this report with respect to previous papers by the authors is the definition of the design strategy, its integration with the formal framework, its implementation within the CAPRIV tool and its application to the electronic toll pricing case study.

6 Conclusion

We have shown in this report, how data minimisation requirements can be integrated in the early design phase of a system and turned into a formal model to make it possible to reason about the proposed solutions. As discussed in Section 2, apart from cases where the service can be delivered without the disclosure of any personal data (which are quite rare), there is usually no absolute notion of personal data minimality. The only solution is therefore 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 Section 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 formal guarantees about the solution, the use of the methodology proposed here provides a key benefit in terms of accountability. Accountability is defined in Article 22 of the current draft of the future Data Protection Regulation [10] 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 report 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 others 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 (see Appendix A for a sketch of the CAPRIV environment).

For the reasons discussed in Section 2, the approach described in this report focuses on archi-

tectures. Ongoing work aims to complete this framework with a formal link between architectures and actual implementations (or protocols). Any implementation consistent with an architecture would then meet the properties of the architecture as defined in this report. Another 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 systematic comparison and classification of solutions presented in the litterature (in the style of [19]) based on formal criteria.

References

[1] Abrial, J.R.: Data semantics. In: Koffeman, K. (ed.) Data Base Management. pp. 1–59. North-Holland (1974)

[2] Abrial, J.R.: The B-Book. Cambridge University Press (1996)

[3] Antignac, T., Le Métayer, D.: Privacy architectures: Reasoning about data minimisation and integrity. In: Mauw, S., Jensen, C.D. (eds.) Security and Trust Management, Lecture Notes in Computer Science, vol. 8743, pp. 17–32. Springer (2014)

[4] Antignac, T., Le Métayer, D.: Privacy by design: From technologies to architectures. In: Preneel, B., Ikonomou, D. (eds.) Privacy Technologies and Policy, Lecture Notes in Computer Science, vol. 8450, pp. 1–17. Springer (2014)

[5] Balasch, J., Rial, A., Troncoso, C., Geuens, C.: PrETP: Privacy-Preserving electronic toll pricing. In: Proceedings of the 19th USENIX Security Symposium. pp. 63–78. Washington DC, USA (Aug 2010)

[6] Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. SEI series in Software Engineering, Addison-Wesley, 3rd edn. (September 2012)

[7] Bjørner, D., Jones, C.B. (eds.): The Vienna Development Method: The Meta-Language. Springer (1978)

[8] Deswarte, Y., Aguilar Melchor, C.: Current and Future Privacy Enhancing Technologies for the Internet. Annales Des Télécommunications 61(3–4), 399–417 (2006)

[9] Diaz, C., Kosta, E., Dekeyser, H., Kohlweiss, M., Girma, N.: Privacy preserving electronic petitions. Identity in the Information Society 1(1), 203–209 (2009)

[10] 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 commitee vote provided by the rapporteur, Commission of European Union (october 2013)

[11] Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.: Reasoning About Knowledge. MIT Press, paperback edn. (jan 2004)

[12] 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. pp. 163–178. SEC'13, USENIX Association, Berkeley, CA, USA (2013)

[13] Garcia, F., Jacobs, B.: Privacy-friendly energy-metering via homomorphic encryption. In: Cuellar, J., Lopez, J., Barthe, G., Pretschner, A. (eds.) Security and Trust Management, Lecture Notes in Computer Science, vol. 6710, pp. 226–238. Springer Berlin / Heidelberg (2011), 10.1007/978-3-642-22444-7_15

[14] Goldberg, I.: Privacy Enhancing Technologies for the Internet III: Ten Years Later, vol. Digital Privacy: Theory, Tech., and Practices, pp. 3–18. Auerbach Publications (2007)

[15] Gürses, S., Troncoso, C., Diaz, C.: Engineering Privacy by Design. Presented at the Computers, Privacy & Data Protection conference (Jan 2011)

[16] Hafiz, M.: A Pattern language for developing privacy enhancing technologies. Software: Practice and Experience 43(7), 769–787 (2010)

[17] Halpern, J.Y., Pucella, R.: Dealing with logical omniscience. In: Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge. pp. 169–176. TARK '07, ACM, New York, NY, USA (2007)

[18] Hoepman, J.H.: Privacy design strategies. In: Cuppens-Boulahia, N., Cuppens, F., Jajodia, S., Abou El Kalam, A., Sans, T. (eds.) ICT Systems Security and Privacy Protection, IFIP Advances in Information and Communication Technology, vol. 428, pp. 446–459. Springer Berlin Heidelberg (2014)

[19] Jawurek, M., Kerschbaum, F., Danezis, G.: Privacy technologies for smart grids - a survey of options. Tech. Rep. MSR-TR-2012-119, Microsoft Research (November 2012)

[20] de Jonge, W., Jacobs, B.: Privacy-Friendly electronic traffic pricing via commits. In: Degano, P., Guttman, J., Martinelli, F. (eds.) Formal Aspects in Security and Trust, vol. 5491, pp. 143–161. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)

[21] Kerschbaum, F.: Privacy-preserving computation (position paper). Presented at the Annual Privacy Forum conference (2012)

[22] Kung, A.: PEARs: Privacy enhancing architectures. In: Proceedings of the Annual Privacy forum. Greece (2014)

[23] Le Métayer, D.: Privacy by design: A matter of choice. In: Gutwirth, S., Poullet, Y., De Hert, P. (eds.) Data Protection in a Profiled World. pp. 323–334. Springer Netherlands (2010), 10.1007/978-90-481-8865-9_20

[24] Le Métayer, D.: Privacy by design: A formal framework for the analysis of architectural choices. In: Proc. of the Third ACM Conference on Data and Application Security and Privacy. pp. 95–104. CODASPY '13, ACM, New York, NY, USA (2013)

[25] 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'07). pp. 115–124 (Jan 2007)

[26] Maffei, M., Pecina, K., Reinert, M.: Security and privacy by declarative design. In: Computer Security Foundations Symp. (CSF), 2013 IEEE 26th. pp. 81–96 (2013)

[27] 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.) Advanced Information Systems Eng. Workshops, Lecture Notes in Business Information Processing, vol. 148, pp. 455–465. Springer Berlin Heidelberg (2013)

[28] Mulligan, D.K., King, J.: Bridging the gap between privacy and design. University of Pennsylvania Journal of Constitutional Law 14(4), 989–1034 (April 2012)

[29] 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 Advances in Information and Communication Technology, vol. 352, pp. 283–296. Springer Berlin Heidelberg (2011)

[30] Popa, R.A., Balakrishnan, H., Blumberg, A.J.: VPriv: protecting privacy in Location-Based vehicular services. In: Proceedings of the 18th USENIX Security Symposium. pp. 335–350. Montreal, Canada (Aug 2009)

[31] Pucella, R.: Deductive algorithmic knowledge. CoRR cs.AI/0405038 (2004)

[32] Rezgui, A., Bouguettaya, A., Eltoweissy, M.Y.: Privacy on the web: facts, challenges, and solutions. Security Privacy, IEEE 1(6), 40–49 (Nov 2003)

[33] Rial, A., Danezis, G.: Privacy-Preserving smart metering. Technical report MSR-TR-2010-150, Microsoft Research (Nov 2010)

[34] Romanosky, S., Acquisti, A., Hong, J., Cranor, L.F., Friedman, B.: Privacy patterns for online interactions. In: Proc. of the 2006 Conference on Pattern Languages of Programs. pp. 12:1–12:9. PLoP '06, ACM, New York, NY, USA (2006)

[35] Spiekermann, S., Cranor, L.F.: Engineering privacy. IEEE Transactions on Software Engineering 35(1), 67–82 (2009)

[36] Sprenger, C., Basin, D.: Developing security protocols by refinement. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. pp. 361–374. CCS '10, ACM, New York, NY, USA (2010)

[37] Troncoso, C., Danezis, G., Kosta, E., Preneel, B.: Pripayd: privacy friendly pay-as-you-drive insurance. In: Ning, P., Yu, T. (eds.) Proc. of the 2007 ACM Workshop on Privacy in the Electronic Society, WPES 2007. pp. 99–107. ACM (2007)

[38] Tschantz, M.C., Wing, J.M.: Formal methods for privacy. In: Cavalcanti, A., Dams, D.R. (eds.) FM 2009: Formal Methods, Lecture Notes in Computer Science, vol. 5850, pp. 1–15. Springer Berlin Heidelberg (2009)

Appendix A: Computer Aided Privacy Engineering Tool

CAPRIV, the computer aided privacy engineering tool, has been developed to help non-experts designers to build architectures and to verify their privacy properties according to the strategy and model presented in this report. The interface and the back-end have been themselves developed with privacy by design in mind. For example, functions are assumed to be invertible by default and 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 notation. This is mainly achieved through the use 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 appendix 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: Require, Design, and Verify. The View part is composed of three other panes: Requirements, Architecture, and Proofs. The View part shows the results of the interactions of the designer with the Model part.

Requirements. 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 indexed or a function is not invertible, can be selected throughout the process. The equations defining the service are then declared (using these elements). Finally, the confidentiality and integrity requirements are defined based on the elements and the service. The current proof of concept version of the tool only supports simple equations (without function application nesting). This limitation can be circumvented by decomposing the service equations in simpler subequations. The Require panel is shown in Figure 2.

Design. The next step for the designer is to build an architecture meeting the requirements. To this aim, CAPRIV implements a design cycle following the strategies presented in Section 2.3. The designer must define the pre-existing constraints before choosing, for each equation in the service, 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 Figure 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 of a computation must themselves be produced in one way or another and be available to the component). The designer can then formally verify the confidentiality and the integrity requirements.

Implementation.

CAPRIV has been developed in Scala[8] which mixes object orientation and (impure) functional programming while relying on an elaborated type system. The program is executed on the Java

[8] Odersky, M., Spoon, L., Bill, V.: Programming in Scala. Artima (2010)

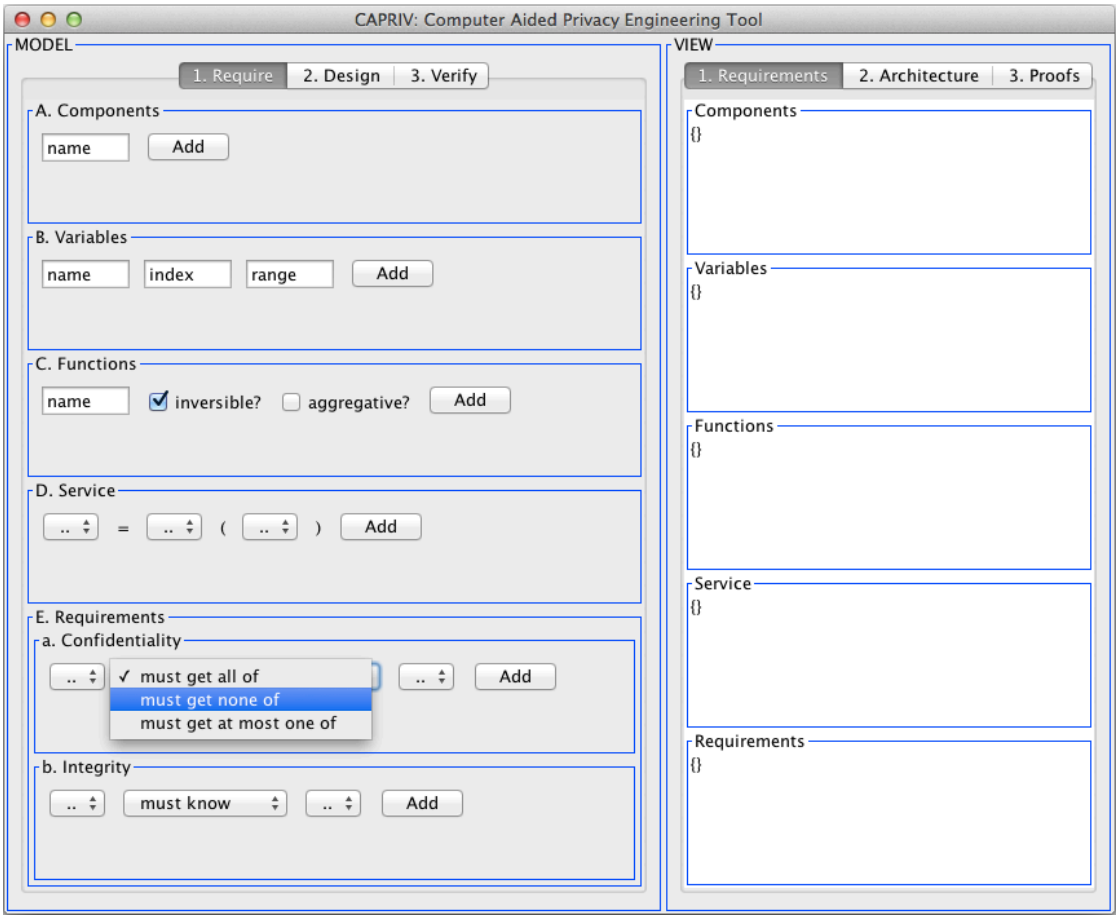


Figure 2: Require panel.

Virtual Machine for platform-independence and compatibility with Java libraries. The use of Scala makes it possible to follow the so-called LCF approach[9] which statically guarantees that theorems can only be built from axioms and inference rules.

Some of the verification features rely on external theorem provers and CAPRIV acts as a frontend interface for the Why3 framework[10]. 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[11] or Z3[12] 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

[9]Gordon, M.J.C.: Proof, Language, and Interaction, chap. From LCF to HOL: a short history. MIT Press (2000)

[10]Bobot, F., Filliâtre, C., J., Marché, C., Paskevich, A.: Why3: Shepherd your herd of provers. In: In Workshop on Intermediate Verification Languages (2011)

[11]Conchon, S., Contejean, E.: The Alt-Ergo automatic theorem prover. `http://alt-ergo.lri.fr/` (2008)

[12]de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, 14th Int. Conf., TACAS 2008. Lecture Notes in Computer Science, vol. 4963, pp. 337340. Springer (April 2008)

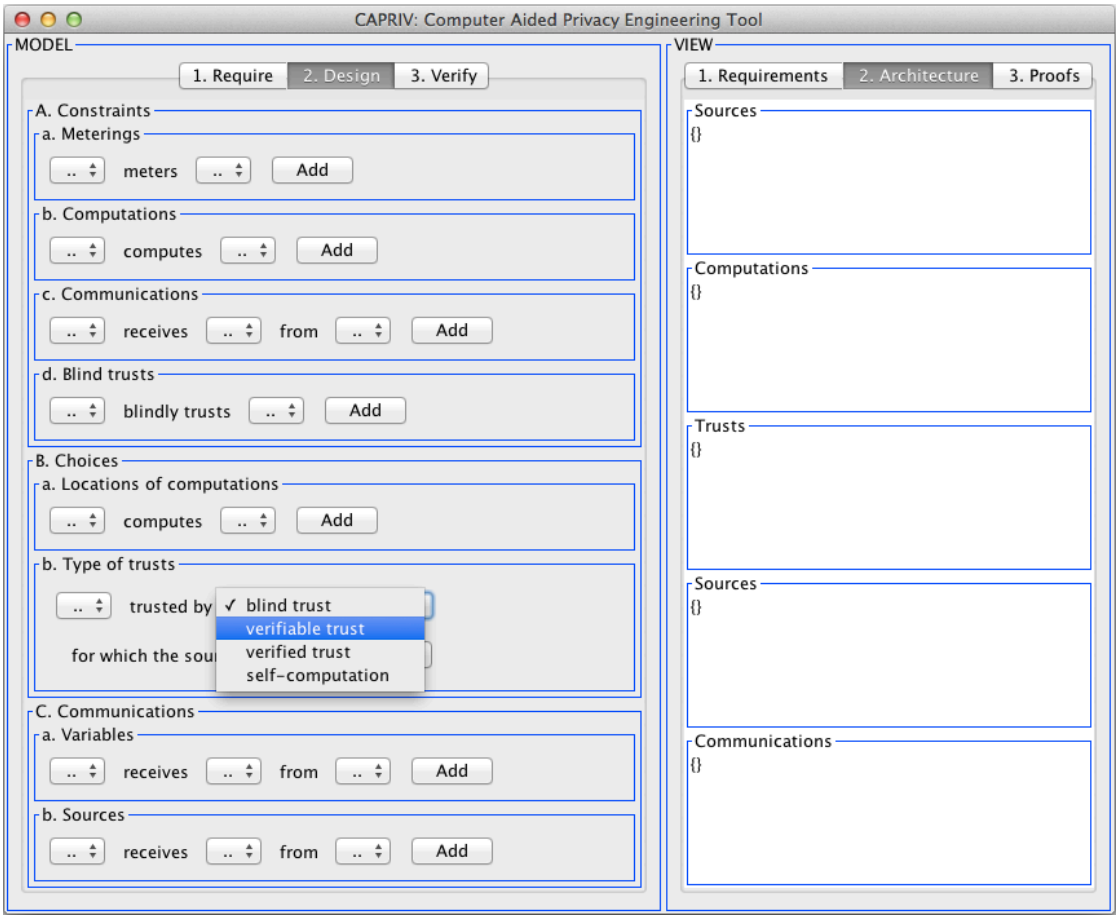
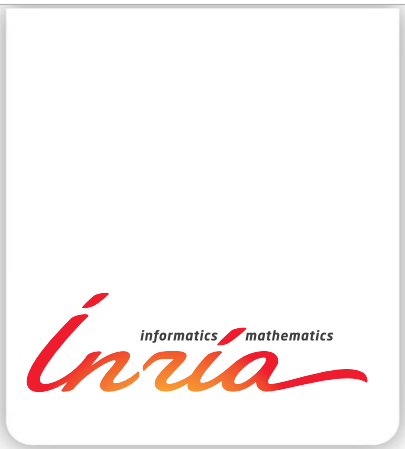


Figure 3: Design panel.

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.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

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

ISSN 0249-6399