Is Privacy by Construction Possible?

Gerardo Schneider

Department of Computer Science and Engineering,
University of Gothenburg, Sweden.
gerardo@cse.gu.se

Abstract. Finding suitable ways to handle personal data in conformance with the law is challenging. The European General Data Protection Regulation (GDPR), enforced since May 2018, makes it mandatory to citizens and companies to comply with the privacy requirements set in the regulation. For existing systems the challenge is to be able to show evidence that they are already complying with the GDPR, or otherwise to work towards compliance by modifying their systems and procedures, or alternatively reprogramming their systems in order to pass the eventual controls. For those starting new projects the advice is to take privacy into consideration since the very beginning, already at design time. This has been known as *Privacy by Design* (*PbD*). The main question is how much privacy can you effectively achieve by using PbD, and in particular whether it is possible to achieve *Privacy by Construction*. In this paper I give my personal opinion on issues related to the ambition of achieving Privacy by Construction.

1 Introduction

Handling *personal data* adequately and in conformance with common sense, standards and regulations is a big challenging task. The already approved European General Data Protection Regulation (GDPR) [20], put into force this year, attempts to protect individuals by putting stringent constraints on how personal data should be collected, stored and treated. Companies and individuals must comply or pay huge fines.

The obvious approach for those developing new software is to consider this important non-functional requirement as part of its design and development. *Privacy by Design* [13] is based on the idea that any personal data processing environment should be designed so that privacy is taken into account as early as the requirement elicitation phase and when designing the software architecture. Given that not all software development is done following a rigorous methodology (i.e., quite often one does not make an explicit design but just write the code having the design "in mind"), the idea would be that privacy should then be taken into account at least when programming. That is, the overall suggestion is that privacy should be as important as the functional aspect of your design and programming task.

This is easier to say than to do. In practice there is a big gap between the technical concepts handled by software architects and the prescriptions stated by

laws and regulations. From the design point of view, software architects usually use different kinds of diagrams at different levels of abstractions in order to make conceptual models and more detailed architectures describing how the software system is to be structured. Many levels of abstraction may be present in such design (if any) including the modeling of basic units, their integration into bigger components and modules, their interconnection, and communication protocols among such elements.

From the programming point of view, programmers mostly focus in the functionality of the software (*what* the software is supposed to do) and eventually on very specific non-functional requirements like performance (execution and data retrieval speed, response time, etc.). That said, most of our software engineers and programmers are not trained in security and privacy, and thus they might not have the necessary background to handle such non-functional requirements. Even if they are aware and knowledgeable in those areas, there is a general lack of tools and methodologies to design and program having privacy in mind. They are not to blame though. Security and privacy threats have been around for a while but only recently became an imperative, not only because of the external (legal) pressure (e.g., GDPR) but also because the number of vulnerabilities, and misuse cases, has been made more apparent due to increased media coverage (e.g., recent scandal with Facebook [5, 10], and the Google case [7]). Complying with new privacy requirements it is not easy though (see for instance an analysis of how Facebook tries to address GDPR issues [17]).

Though each user has an own understanding on what is personal and sensitive data, it is in general not always clear what defines the privacy frontiers. Independently of the individuals' perspective on their own data[1] we are here concerned with privacy from the legal point of view. That is, what is required by standards and regulations like the already mentioned GDPR. The law is generally written in a normative way stating the rights and obligations of citizens and corporations in different circumstances. The GDPR defines what personal data is and establishes constraints in the way data may be collected, used and stored. Such regulations and principles, written in natural language, need to be interpreted and implemented by software engineers into technical designs and concrete software that are required to be compliant with the privacy legislation. One way to do this would be to follow the PbD philosophy, and in particular we could aim at a systematic and (semi-)automatic way of achieving that, i.e. achieving privacy *by construction*.

X-by-Construction approaches may be defined as *"a step-wise refinement process from specification to code that automatically generates software (system) implementations that by construction satisfy specific non-functional properties [...]"*.[2] In this paper we assume the premise that it could be desirable to achieve *Privacy by Construction*, that is the possibility to automatically refine (formal)

[1] Technically, the individual or companies we are taken data from/about are called *data subjects*, and those handling the data are the *data controllers*.

[2] http://fmt.isti.cnr.it/ mtbeek/ISoLA18.html.

specifications so the generated code satisfies the privacy properties under consideration.

Privacy cannot be enforced uniquely by technical means and requires to be seen from a multidisciplinary perspective, including computer science, computer security and cryptography, law, social sciences and economics [18]. In this paper we focus only on privacy from a computer science perspective, starting in particular from Privacy by Design (PbD)[3], and we discuss about the feasibility of achieving *Privacy by Construction*.

The paper is organized as follows. In next section we give a very brief overview of some key privacy concepts from the GDPR. Section 3 recalls the main ideas behind PbD and gives an incomplete account of the state of the art in the area, and provides some examples for achieving privacy *a posteriori*. Section 4 presents a personal point of view on what may be achieved with PbD, and list some limitations and challenges for specific privacy issues concerning PbD and Privacy by Construction. Finally, we conclude in the last section with a general discussion.

2 GDPR

The European *General Data Protection Regulation* (GDPR) [21] is a document containing around 90 articles whose aim is to regulate *personal data* processing. The text replaces, among others, the Directive 95/46/EC of the European Parliament, and improves over the original proposal given in 2012 [20]. As any text of this kind, the regulation is complex and full of definitions.

For the purpose of this paper we are only interested in naming that the regulation talks about seven *personal data processing principles*, namely i) Lawfulness, fairness and transparency; ii) Purpose limitation; iii) Data minimization; iv) Accuracy; v) Storage limitation; vi) Integrity and confidentiality; vii) Accountability. Besides the above, the GDPR stipulates that data subjects have several *rights* including i) Right to information, access and rectification of personal information; ii) Right to object to personal data processing; iii) Right not to be subject to automated individual decisions; iv) Right to be forgotten (concerning data erasure); v) Right to data portability. Moreover, Article 6 of the regulation stipulates six *lawful grounds* for data processing operations, namely: i) Consent; ii) Performance of a contract; iii) Compliance with a legal obligation; iv) Protection of vital interests; v) Public interest; vi) Overriding interest of the controller.

Besides the original document, it is worth reading the critical review of the regulation (and on the comparison with previous versions) done by de Hert and Papakonstantinou [25].

[3] In the rest of the paper we will use the acronyms PbD for Privacy by Design.

3 Privacy by Design

In this section we briefly give an overview of PbD, we discuss some issues concerning the achievement of PbD, and we provide some examples following the approach. We finish with some examples of *a posteriori* techniques for very specific aspects of privacy.

3.1 What is PbD?

In a nutshell and informally, *Privacy by Design* (PbD) is an approach for ICT project development that promotes taking privacy and data protection compliance from the very beginning. The concept was introduced by Ann Cavoukian[4] in the 90's and further developed over the following years. In [12] Cavoukian describes the following seven foundational principles for PbD: i) Proactive not reactive, preventative not remedial; ii) Privacy as the default setting; iii) Privacy embedded into design; iv) Full functionality: positive-sum, not zero-sum; v) End-to-End Security: full lifecycle protection; vi) Visibility and transparency: keep it open; vii) Respect for user privacy: keep it user-centric. (See [12] for more details).

The concept was taken by several researchers in the last ten years or so mostly due to the increasing problem associated with Big Data, and more recently due to the GDPR and other strong regulations.

In December 2014 Danezis et. al. [18] wrote a report for the European Union Agency for Network and Information Security (ENISA) in order to *" promote the discussion on how PbD can be implemented with the help of engineering methods"* and to *"[...] provide a basis for better understanding of the current state of the art concerning privacy by design with a focus on the technological side"*.

A more telling explanation of the scope of PbD is given in such report: *"[...] privacy by design is neither a collection of mere general principles nor can it be reduced to the implementation of PETs.*[5] *In fact, it is a process involving various technological and organisational components, which implement privacy and data protection principles. These principles and requirements are often derived from law, even though they are often underspecified in the legal sources."*

The report is very clear on that PbD is still in infancy[6] and makes concrete recommendations for taking the next step in engineering PbD (cf. [18, Section 5.2]). See also [23, 24, 37] concerning the engineering of privacy.

3.2 How to achieve PbD?

From the design point of view, it could be great if software engineers do not need to worry too much about privacy and may only concentrate on the functional

[4] Former Information & Privacy Commissioner of Ontario (Canada).
[5] PET: Privacy-Enhancing Technologies.
[6] The report is from 2014, but to the best of our knowledge the advances in the area have not yet produced mature tools as to be used by industry.

aspect, while tool-assisted design methodologies automatically add the needed privacy checks to the design. This is an ideal situation and some privacy aspects might be handled in this way. Ideally, software engineers will design their software with the tools they are already familiar with and after pressing a "convert" button they would get an enhanced model containing specific privacy checks in some parts and proposals or warning for others. Those models/diagrams would be then edited by a privacy engineer who would validate or modify such checks and add meta-annotations on the model so programmers get more precise hints on how to implement those checks.

That would of course be the first step. A design needs to be synthesized (materialized) into programming code that effectively implements the (privacy-compliant) design. How to ensure such transformation is *sound* (correct) and *complete* (it does address all the privacy principles under consideration)? Formal methods would help to prove soundness. Completeness is more difficult but it could be achieved with respect to a small set of predefined privacy concepts.

A complementary (ideal) solution, from the programming point of view, would be to have specialized libraries offering an API for handling personal data. These libraries would not only ensure that data have been collected with the consent of the user, that it has a time-stamp on when it was collected and eventually a timeout (for retention time). The data would carry the history (its *provenance* as well as information on where it was sent and stored). Ideally, each operation on the data would be checked against the consented purpose, only if the retention time has not expired.

3.3 Some proposals following PbD

PbD has been applied to specific real(istic) cases, including electronic traffic pricing (ETP), smart metering, and location based services (see [29] and references therein). In [29] Le Métayer proposed a framework to express different parameters to take into account for making a choice when defining software architectures, and applied the framework to an ETP.

According to [1] previous work has "focused on technologies rather than methodologies and on components rather than architectures". In that paper, Antignac and Le Métayer moves forward towards an application of PbD to the "architectural level and be associated with suitable methodologies", proposing a more formal approach. In particular they use a *privacy epistemic logic* as a way to specify and prove privacy properties about software architectures.

In [3, 4] Antignac et al. propose an approach based on model transformations, which guarantees that an architectural design, originally focused on functional data requirements, is enhanced with certain privacy principles. In particular, the proposal is to automatically transform data flow diagrams (DFDs) into so-called *privacy-aware data flow diagrams* (or PA-DFDs for short). PA-DFDs are DFDs enhanced with a log system (for accountability), and suitable checks concerning retention time and purpose for each operation on sensitive data (storage, forwarding, and processing of data). The software architect then focuses only on defining the function part as DFDs, and then check and eventually extend the

obtained PA-DFD. The ultimate goal would be to automatically get a program template from the PA-DFD, helping the programmer to identify in which points to perform suitable privacy checks. The approach has not been implemented so it remains to check whether it is feasible, it captures relevant privacy checks, and it scales.

More recently Basin et al [6] proposed a methodology for auditing GDPR compliance by using *business process models*. One of the key insights of the paper is to identify "purpose" with a "process". Besides, it shows how to automatically generate privacy policies from the model and detect violations of data minimization, and gives arguments on why GDPR compliance cannot be entirely automated.[7]

A different, and complementary, way of viewing PbD is to consider privacy even at a higher level of abstraction, not from the architectural point of view but rather from the meta-level. This is the line of work proposed by Hoepman on the definition of privacy *design strategies* [26]. These strategies are to be taken into account during the analysis and requirements engineering phase, before designing the software architecture. Colesky et al. [16] suggest an additional level of abstraction between privacy design strategies and privacy patterns by considering *tactics*. Notario et al. [30] present a methodology for engineering privacy based on existing state-of-the-art approaches like PIA (Privacy Impact Assessment) and risk management, among others. These approaches, though valuable in the context of PbD, are even further from the implementation and they "suffer" from the same issues as for PbD in what concerns its relatedness to the implementation.

Finally, it is worth mentioning the work by Schaefer et al. [35], presented in this very same track of ISoLA'18, on the definition of rules for achieving *Confidentiality-by-Construction*. The approach is based on replacing functional pre-/postcondition specifications (Hoare triples, as traditionally used for classical functional correctness) with confidentiality specifications listing which variables contain secrets. The approach is promising. That said, this is an ongoing work so it remains to see whether it is feasible in practice.

3.4 A posteriori techniques for privacy

Though not the main objective of this paper, we mention here a few attempts to ensure compliance to some of the privacy principles, not by following PbD but rather by do *a posteriori* analysis (once the software has already been developed, both before and after deployment).

In [22] Ferrara and Spoto suggest the use of tainting and other static analysis techniques to detect the potential leak of private sensitive information, combined with abstraction techniques so the results are shown differentially to the players interested in privacy. Despite the generality of the title, the ideas of the paper

[7] Some of the arguments of our paper are very much along the same line as the ones presented in [6]. In particular, the identification of the difficulty to represent purpose at the programming language level.

are applicable to compliance w.r.t. very specific privacy aspects. Note that the static analysis of such properties is undecidable in general.

In [2] Antignac et al. provide a characterization of the data minimization principle in a restricted setting: deterministic (functional) programs and for a notion of minimization based only on the *collection* of data (not on data *processing*). The paper contains results for both the *monolithic* case (when the input comes from only one source) and for the *distributed* case (when the input is collected from multiple independent sources). A technique based on symbolic execution and SAT solvers is presented for computer a *minimizer*, that is a pre-processor that filters the input so data minimization is guaranteed. The main limitations are that computing the minimizer is undecidable in general, that the source code is required (it is white box) and that it only considers a very restrictive version of data collection (the technique only works for the case when minimization is achieved when restricting the input domain, and it should be generalized for more complex notions).

A black box approach for monitoring whether a give program satisfies (strong) data minimization has been proposed in [33, 34] (under the same definition given in [2]). Giving a final verdict (at runtime) that a given program minimizes data is in general not possible, but if the program is non-minimal then the runtime monitor might be able to detect that (no completeness results are given, but if the current, or any past, execution of the program violates data minimization, the monitor would detect it). Under very strict conditions (finiteness of the input domain), it is possible to effectively determine using off-line monitoring if the program satisfies data minimization or not, and in the latter it is possible to extract a minimizer (always using a black box technique). Monitoring a more general (weaker) version of data minimization is not possible in general as it is a more complex hyperproperty, lying in a fragment of HyperLTL known not to be monitorable (it is a "$\forall\forall\exists\exists$" property). This is also the case for a negation of the property. That said, recent ideas on combining static and runtime verification might be useful to help getting a monitor for the negation of the property [8].

4 Is Privacy by Construction Possible?

In the previous section we discussed, at a high level, how PbD may be used at design time, hinted on the possibility of integrating privacy at the programming level, gave some links to the literature on the use of PbD in concrete scenarios as well as some examples of achieving privacy *a posteriori*. The question is still how much privacy may be achieved in practice by PbD. In other words, if you follow a PbD approach how much privacy is really "propagated" into the running software so it could be considered privacy compliant (with for example the GDPR)? Is that possible at all? For which principles? Can we effectively characterize what is achievable by PbD and what is not? Even more, how much of that maybe done *automatically*, achieving then Privacy by Construction?

In this paper we will not be able to answer all the above questions but will rather focus on some aspects. In particular, in this section we list some general

limitations of PbD, and we present challenges concerning few specific privacy issues. Note that the challenges we mention are not necessarily in correspondence with the listed limitations.

4.1 Some limitations

Section 5.1 of the ENISA report [18] lists a number of limitations of PbD. We will not enumerate them all here but will just mention few keys points (see details and examples in the report).[8]

- One important issue is the lack of *compositionality*: a system satisfying a given privacy property is not guaranteed to do so when composed with another one. So, even when we might achieve privacy by using PbD on given components, it is not clear that this could propagated to the whole system.
- There currently is no way to measure how private a system is, nor to compare whether a system is more private than another one: there is lack of suitable privacy *metrics*. So, it is not easy to make a good privacy risk assessment or to assign any kind of economic value to (lack of) privacy.
- There is a trade-off between privacy and utility, so the designer needs to take this into account when taking a PbD approach. The risk is that if privacy takes too much importance at design time, it might jeopardize the utility of the developed software.
- There is a lack of design methodologies and tools, integrated in the software development cycle, facilitating a practical use of PbD.

The above thus gives already a good hint that PbD, by itself, cannot in general guarantee Privacy by Construction; we elaborate on some of the above and other difficulties to achieve it.

Our first (obvious) observation is that by following PbD we run, at least, into the very same problems as when you develop software from (in)formal functional requirements and specifications. Unless you automatically (and with a proof of correctness) translate (*synthesize*) your program, you can only assert with a limited degree of confidence that your software is correct *by construction*. This is because most of the time your design (model or specification) is a different creature from the implemented software, and in most cases the step from your design (model or specification) into the software that will effectively be running in your computer is manual. This step may introduce bugs, or your model (or specification) might be partial, incomplete, or too abstract.

So following PbD will not, by itself and in general, guarantee a private by construction software as new techniques for (automatic) refinement should be developed, and this might be more difficult for privacy as it is for functional properties. That said, for very specific privacy aspects it might be possible to achieve the intended goal. For instance, we could envisage similar techniques as

[8] The report stresses a few times the privacy is much wider than technology (social, legal, political, etc.) but the focus on the discussion here is on the (software-based) technological side only.

the one proposed in [35] for confidentiality. The full feasibility and applicability of this approach is still to be assessed though.

That said, in our opinion it is strongly recommended to use PbD given that you may at least ensure that you have taken (certain) privacy aspects into consideration when you design your software, expecting that in the development phase such considerations are followed and applied so they are reflected in the final product.

4.2 Some challenges

We summarize in what follows some of the key challenges in achieving Privacy by Construction, and we list possible ways to handle some of those. Obviously technology cannot help solve all the problems, but even if we were to limit our analysis only to those aspects related to the technical, we would not be able to cover all of them in this paper. Instead of focusing on how to deal with specific privacy principles or properties, we take our discussion based on some of the key concepts in the GDPR. In particular, we consider one principle (*purpose*), one right (the *right to be forgotten*), and two lawful grounds (*consent* and *compliance*).[9]

Purpose Today's programming languages do not give good support to represent purpose as required by the GDPR, maybe with the exception of having an enumerative type with the different purposes the service might use the data. This, on the other hand, would not ensure by itself that the data will be indeed used only for the intended purpose. For that there might be a need to define specialized libraries, with carefully defined abstract data type *purpose* (or defined special programming language primitives), guaranteeing that only certain operations may be applied to the data according to the consented purpose. That said, it will not be easy to take care of this if one allows for the data to navigate to non-controlled parts of the program, or if it is processed in a way that the sensitive (*high*) data is propagated through explicit and implicit flows to public (*low*) data. So, it will be difficult to give strong guarantees as it may be difficult to ensure the programmer uses such library operations or built-in constructs. In order to handle this and do something "by construction" one possibility would be to define new data types where each piece of data comes with its purpose (and other relevant privacy information). Very much like "sticky policies" [31], that are conditions and constraints (*policies*) attached to data that describe how it should be treated. These policies would travel with the data so they (ideally) are enforced whenever the data is to be used.

If one follows a PbD approach, it might be possible to include purpose at the design level, but it is not clear how this could be translated into the real implementation. For instance, some of the approaches mentioned in Section 3.3 envisage the declaration of the purpose in different ways. Basin et al. [6] proposes that business processes explicitly represent one or more purposes, by associating

[9] The reader may want to see [36, 40] for a different perspective on challenges in PbD.

a "process" with a "purpose". They propose a methodology for auditing GDPR based on proving a correspondence between GDPR, the business process representation and the implementation. It is not clear, however, how this could be done automatically. In particular, the relation between the business process and the implementation is very much the same that exists today between any (functional) formal specification and an implementation. Similarly Antignac et al. [3, 4] propose to have purposes just as a meta-data associated with the data, and then getting a PA-DFD that would add a check before using the data on whether the intended use is for the consented purpose. Though not presented in the papers, the idea would be that the programmer gets a skeleton of a program (automatically synthesized from the PA-DFD) manually adding the concrete privacy checking during implementation (at design time, the purpose is just a string with no concrete meaning except its belonging to an enumerative set).

Finally, a last example on how to deal with purpose is the one discussed in [2] in the context of data minimization. In such papers the "purpose" is defined by the program itself. This, however, is a quite simplistic way of handling purpose and would only work for limited number of applications handling data for very specific purposes.

As a final comment, the issue of purpose is very much related to the general problem of data *usage*, so it could be interesting to investigate whether techniques from *usage control* could be used in this context (see for instance [28] and references therein). Another interesting line of work, and probably more relevant than usage control, in relation to purpose is that of *purpose-based access control* [9], and other similar work, whose aim is to design access control mechanisms based on purpose. See the related work section in [6] for an overview of these and other access control variants aiming at capturing the notion of purpose.

Right to be forgotten and retention time In order to enforce data deletion whenever the data subject requires it, or whenever there is a timeout, seems very difficult. This would require to keep very advanced techniques to track data provenance [14]. One way to approach this issue could be to follow a technique based on information-flow security, in particular the one developed for *information erasure* [15]. In [39] Del Tedesco et al. use dynamic taint analysis to track how sensitive data propagate through a program and erase them on demand. The method is implemented as a library for Python. The approach is further developed in [38] Hunt el al. were it is presented a framework to express a rich class of erasure policies. Similar approaches could be used for the GDPR rights, but the main issue still remains: the above would only work if the date remains inside one specific software program. As soon as the data is forwarded somewhere else (or saved on an external device) it should be thus tracked and we do not know today how to solve this.

Another idea would be to use *stick policies* [31] (cf. purpose above). As today, this is a nice theoretical concept but it lacks practical implementations. As for the erasure papers mentioned above, implementing sticky policies for one specific software might be feasible, but still the challenge is to make it work across different applications and platforms.

Consent The problem of consent seems to be a bit easier to handle if the focus is only on asking permission from the data subject to collect certain data for a given purpose: it only suffices to always add a standard question whenever the data controller collects data and only do so if the data subject agrees. So, consent might be easy to implement whenever there is an explicit collection of data where the data subject is directly asked before the collection of data. This, however, it is an oversimplification.

First, we know that if consent is asked for every explicit collection of data and for approval of specific policies (e.g. *terms of service*) the user will simply agree without really knowing what is she giving consent to.[10]

Second, sometimes data is collected in a continuous manner in ways that makes it extremely difficult to ask for consent. Think about street cameras and broadcasting of wi-fi devices, just to mention a couple of ways of getting personal data without explicit consent. An example of this is the data collection performed by Google Street View cars: "*It is now clear that we have been mistakenly collecting samples of payload data from open wifi networks, even though we never used that data in any Google products*" [27]. The software should be selective on which wi-fi data to collect, but is this possible? A possible solution would be to develop a specialized software that runs in a protected environment (enclave) where the collected data is stored and analyzed in a "smart" way so only what is relevant is forwarded outside the enclave for further processing. How to implement this "smart" solution?

The position paper [11] discusses an approach based on the use of registries to deal with consent in the context of the Internet of Things. It could be interesting to see how this solution could be generalized.

Compliance The notion of compliance is very much related to purpose [6], and as we have discussed above including the notion of purpose as a first class citizen in programming languages is still a challenge. So, giving a technical means to have privacy compliance "by construction" will only be possible whenever all the other issues are solved, and even then it might be very difficult if not impossible. This is the case since in order to be able to enforce compliance might require to have solutions that are enforced by the runtime execution environment in many different platforms.

Think of a photo that is collected for the purpose of helping researchers in image processing. Let us assume that the photo comes with a sticky policy explicitly specifying that the purpose is "research", that it should not be forwarded to any other data controller, and that the retention time (the photo should be erased from any storage device) is two weeks. In order to enforce this, in a relatively easy way, the photo should be received via an application (via suitable APIs) and only handled by that application. If so, we could save the photo and make all needed manipulations in a safe enclave (very much like a sandbox)

[10] The GDPR [21] is very explicit on that consent should be given *freely*, in an *informed and unambiguous* way, and that it should *cover all processing activities carried out for the same purpose*. A separate consent should be given for each separate purpose.

where only the so-called research engine would be running. If we cannot guarantee that every single device and platform contains the needed enforcement mechanism, we cannot make any strong claim. The photo might be uploaded to a different application, or stored in a disk and then forwarded to somewhere else (there is no way to enforce the sticky policy unless the software associated with handling the photo has the mechanisms to do so).[11]

Some researchers believe that we should not worry very much about trying to enforce compliance "by construction" but rather leave the burden to data controllers: it is on their own interest to be able to give evidence that they do comply with the law whenever they are challenged to do so. This, however, might seem an attempt to solve the problem by avoiding it, or even by assuming it is not a problem. Not quite. The idea is to provide as much support as possible to data controllers (e.g., in the form of guidelines, and programming libraries as discussed earlier) to handle data according to the law. If they do not do that (e.g., do not use the specialized libraries) and misuse the data (e.g., use them for a different purpose than the one consented by the data subject) it is their problem as they will have to prove that they are not guilty.

5 Final Discussion

Achieving privacy compliance is not easy. Many solutions should be addressed at the level of organizational procedures and practices, and others by technical means (software-based solutions). Guaranteeing most privacy principles in already existing systems is in most cases impossible, and if possible then extremely difficult and costly. When building new systems, following a PbD approach is definitively a step forward but not a panacea. More research is needed in order to be able to advance the state-of-practice in this domain. Achieving perfect Privacy by Construction is, in my opinion, impossible in general and extremely expensive in the best case. That said, the fines for non-compliance are extremely high[12] so investment in privacy solutions is a must.

Another way to think about privacy compliance is to provide the technical means to enhance data with *sticky* privacy policies containing retention time, purpose, etc. and ask the data controller to respect and enforce the data subject's policy. That is, it is up to the data controller to prove that its handling of data complies with the data subject's policy and the law. In this approach, the

[11] An example of such a (limited) mechanism is given in [32] for photo sharing in social networks by using a combination of sticky policies with attribute-based encryption. The mechanism works by encrypting parts of the picture so only allowed users can see what they are supposed to, but if somebody has permission to download the picture to the local disk, there is no way to enforce the sticky policy after that. Since the enforcement mechanism (encryption/decryption, permission checking, etc.) is only done in a particular application platform (Diaspora [19]), the user could forward the decrypted picture to anyone else if she has permission to download it.

[12] It is stipulated that non-compliance might imply fines up to €20 million or 4% of the annual turnover of the company.

controlling agencies do not try to ask for proofs of absence of privacy breaches *a priori*, but if there is any privacy leak then it should be detected *a posteriori* by suitable audits. The burden is thus upon the data controller on choosing the best solution to avoid paying for non-compliance.

A notion left out in this paper has been the notion of *adversary* (or *attacker model*). This has been intentional as the objective of this paper is not to give an exhaustive overview of the field nor to propose solutions but rather to give a personal point of view on the issue of Privacy by Construction. That said, any feasible solution should provide a proper attacker model as well as proofs (or at least convincing evidence) that the proposed approach preserves privacy with respect to those attackers. If possible, it should also make explicit what are the assumptions in order to make it easier to detect potential vulnerabilities (claims in only one direction usually hide some unwanted side effects that could compromise privacy). Besides, we have not been able to cover many other interesting and very relevant aspects of privacy, including anonymization, accountability, unlinkability, transparency, intervenability, and many other concepts present in the GDPR.

All the statements and claims done in this paper reflects a personal point of view, not based on experimental nor on theoretical (im)possibility results. I would like to be proved wrong on my pessimistic views and right on those more optimistic opinions. Hope for new advancements in privacy research so we can all achieve a good trade-off between privacy and utility+transparency.

Acknowledgements I would like to thank Daniel Le Métayer for his valuable comments on an early draft of this paper, and Thibaud Antignac for all the fruitful discussions we have had on privacy by design. This research has been partially supported by the Swedish Research Council (*Vetenskapsrådet*) under grant Nr. 2015-04154 (*PolUser: Rich User-Controlled Privacy Policies*).

References

1. Antignac, T., Le Métayer, D.: Privacy by design: From technologies to architectures - (position paper). In: APF'14. LNCS, vol. 8450, pp. 1–17. Springer (2014)
2. Antignac, T., Sands, D., Schneider, G.: Data Minimisation: A Language-Based Approach. In: IFIP Information Security & Privacy Conference (IFIP SEC'17). IFIP Advances in Information and Communication Technology (AICT), vol. 502, pp. 442–456. Springer Science and Business Media (2017)
3. Antignac, T., Scandariato, R., Schneider, G.: A Privacy-Aware Conceptual Model for Handling Personal Data. In: 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation – ISoLA'16 (1); Track: Privacy and Security Issues in Information Systems. LNCS, vol. 9952, pp. 942–957. Springer (2016)
4. Antignac, T., Scandariato, R., Schneider, G.: Privacy Compliance via Model Transformations. In: International Workshop on Privacy Engineering (IWPE'18) at IEEE EuroS&P Workshops. pp. 120–126. IEEE (2018)

5. Aziza, B.: Facebook privacy scandal hearings: What you missed. Appeared at Forbes online. Accessed on May 16, 2018. Available at https://www.forbes.com/sites/ciocentral/2018/04/16/facebook-privacy-scandal-hearings-what-you-missed/#9a41af57ab9c (April 2018)
6. Basin, D., Debois, S., Hildebrandt, T.: On Purpose and by Necessity: Compliance under the GDPR. In: Twenty-Second International Conference on Financial Cryptography and Data Security (2018), to appear
7. BBC News: Google loses 'right to be forgotten' case. Accessed on April 14, 2018. Available at http://www.bbc.com/news/technology-43752344?SThisFB (April 2018)
8. Bonakdarpour, B., Sanchez, C., Schneider, G.: Monitoring Hyperproperties by Combining Static Analysis and Runtime Verification. In: 8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation – ISoLA'18; Track: A Broader View on Verification: From Static to Runtime and Back. LNCS, Springer (2018), to appear
9. Byun, J., Bertino, E., Li, N.: Purpose based access control of complex data for privacy protection. In: 10th ACM Symposium on Access Control Models and Technologies (SACMAT'05). pp. 102–110. ACM (2005), http://doi.acm.org/10.1145/1063979
10. Cadwalladr, C., Graham-Harrison, E.: Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. Appeared at The Guardian. Accessed on May 16, 2018. Available at https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election (March 2018)
11. Castelluccia, C., Cunche, M., Le Métayer, D., Morel, V.: Enhancing transparency and consent in the iot. In: EuroS&P Workshops'18. pp. 116–119 (2018)
12. Cavoukian, A.: Privacy by design: The 7 foundational principles (2009)
13. Cavoukian, A.: Privacy by design: Origins, meaning, and prospects. Privacy Protection Measures and Technologies in Bus. Org.: Aspects and Standards 170 (2011)
14. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Found. Trends databases 1(4), 379–474 (2009)
15. Chong, S., Myers, A.C.: Language-based information erasure. In: Proceedings of the 18th IEEE Workshop on Computer Security Foundations. pp. 241–254. CSFW '05, IEEE Computer Society (2005)
16. Colesky, M., Hoepman, J., Hillen, C.: A critical analysis of privacy design strategies. In: IEEE Security and Privacy Workshops. pp. 33–40. IEEE Computer Society (2016), http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7517741
17. Constine, J.: A flaw-by-flaw guide to facebook's new gdpr privacy changes. Available at https://techcrunch.com/2018/04/17/facebook-gdpr-changes (April 2018)
18. Danezis, G., Domingo-Ferrer, J., Hansen, M., Hoepman, J.H., Le Métayer, D., Tirtea, R., Schiffner, S.: Privacy and data protection by design. ENISA Report (January 2015)
19. Diaspora: Diaspora. https://joindiaspora.com (2016)
20. European Commission: Proposal for a General Data Protection Regulation. Codecision legislative procedure for a regulation 2012/0011 (COD), European Commission, Brussels, Belgium (January 2012)
21. European Commission: General Data Protection Regulation (GDPR). Regulation 2016/679, European Commission, Brussels, Belgium (April 2016)
22. Ferrara, P., Spoto, F.: Static analysis for GDPR compliance. In: ITASEC'18. CEUR Workshop Proceedings, vol. 2058. CEUR-WS.org (2018)

23. Gürses, S., Troncoso, C., Diaz, C.: Engineering privacy by design (2011)
24. Gürses, S., Troncoso, C., Diaz, C.: Engineering privacy by design reloaded (2015)
25. Hert, P.D., Papakonstantinou, V.: The new general data protection regulation: Still a sound system for the protection of individuals? Computer Law & Security Review 32(2), 179–194 (April 2016)
26. Hoepman, J.: Privacy design strategies - (extended abstract). In: IFIP-SEC'14. IFIP Advances in Information and Communication Technology, vol. 428, pp. 446–459. Springer (2014)
27. Kiss, J.: Google admits collecting wi-fi data through street view cars. The Guardian. Available at https://www.theguardian.com/technology/2010/may/15/google-admits-storing-private-data (May 2010)
28. Lazouski, A., Martinelli, F., Mori, P.: Usage control in computer security: A survey. Computer Science Review 4(2), 81–99 (2010)
29. Le Métayer, D.: Privacy by design: a formal framework for the analysis of architectural choices. In: CODASPY'13. pp. 95–104. ACM (2013)
30. Notario, N., Crespo, A., Kung, A., Kroener, I., Le Métayer, D., Troncoso, C., del Álamo, J.M., Martín, Y.S.: PRIPARE: A new vision on engineering privacy and security by design. In: Cyber Security and Privacy (CSP'14). Communications in Computer and Information Science, vol. 470, pp. 65–76. Springer (2014)
31. Pearson, S., Mont, M.C.: Sticky policies: An approach for managing privacy across multiple parties. IEEE Computer 44(9), 60–68 (2011)
32. Picazo-Sánchez, P., Pardo, R., Schneider, G.: Secure Photo Sharing in Social Networks. In: IFIP Information Security & Privacy Conference (IFIP SEC'17). IFIP Advances in Information and Communication Technology (AICT), vol. 502, pp. 79–92. Springer Science and Business Media (2017)
33. Pinisetty, S., Antignac, T., Sands, D., Schneider, G.: Monitoring data minimisation. Tech. rep. (2018), http://arxiv.org/abs/1801.02484
34. Pinisetty, S., Sands, D., Schneider, G.: Runtime Verification of Hyperproperties for Deterministic Programs. In: 6th Conference on Formal Methods in Software Engineering (FormaliSE@ICSE'18). pp. 20–29. ACM (2018)
35. Schaefer, I., Runge, T., Knüppel, A., Cleophas, L., Kourie, D., Watson, B.W.: Towards Confidentiality-by-Construction. In: 8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation – ISoLA'18; Track: X-by-Construction. LNCS, Springer (2018), to appear
36. Spiekermann, S.: The challenges of privacy by design. Commun. ACM 55(7), 38–40 (2012), http://doi.acm.org/10.1145/2209249.2209263
37. Spiekermann, S., Cranor, L.F.: Engineering privacy. IEEE Trans. Software Eng. 35(1), 67–82 (2009)
38. Tedesco, F.D., Hunt, S., Sands, D.: A semantic hierarchy for erasure policies. In: ICISS'11. LNCS, vol. 7093, pp. 352–369. Springer (2011)
39. Tedesco, F.D., Russo, A., Sands, D.: Implementing erasure policies using taint analysis. In: NordSec'10. LNCS, vol. 7127, pp. 193–209. Springer (2010)
40. Tsormpatzoudi, P., Berendt, B., Coudert, F.: Privacy by design: From research and policy to practice - the challenge of multi-disciplinarity. In: Third Annual Privacy Forum on Privacy Technologies and Policy (APF). LNCS, vol. 9484, pp. 199–212. Springer (2015)