Mechanizing Privacy by Design

David Basin*
basin@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

François Hublet
francois.hublet@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Srđan Krstić
srdan.krstic@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Hoàng Nguyễn
hoang.nguyen@inf.ethz.ch
ETH Zurich
Zurich, Switzerland

Abstract

Privacy by design requires integrating data protection into systems from the outset, during their design, rather than building it in later. Related legislation does not specify how to achieve this and mainstream languages and frameworks lack support for privacy by design. To address this long-standing problem, we have developed different, effective technical solutions. First, we have developed powerful logic-based tools that enforce formal data protection policies at runtime by controlling relevant system actions. Second, we have proposed methods and tools for integrating privacy models into system design models, enabling model-driven privacy enforcement. We report on our methods, tools, and practical experiences using them.

CCS Concepts

• **Software and its engineering → System modeling languages**; • **Security and privacy → Software security engineering**; **Access control**.

Keywords

Data protection, Runtime enforcement, Model-driven development

ACM Reference Format:
David Basin, François Hublet, Srđan Krstić, and Hoàng Nguyễn. 2025. Mechanizing Privacy by Design. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3719027.3748271

1 Introduction

The right to privacy is enshrined in national and supranational law. For example, the GDPR [10] imposes comprehensive data protection requirements on all entities operating in the EU. In particular, it stipulates that personal data may be collected only for specific, explicit, and legitimate purposes [10, Art. 5(1)(b)], and processed only with user consent or another valid legal basis [10, Art. 6(1)]. Organizations processing personal data cannot ignore these laws or the penalties for non-compliance. This leads to a crucial technical challenge: *how to build systems that comply with these laws.*

Existing regulations and standards require appropriate technical measures to be taken to ensure compliance, but offer limited guidance on concrete solutions. For example, the GDPR mandates 'data protection by design and by default' [10, Art. 25], known as 'privacy by design' [8], which emphasizes integrating data protection and auditing from the start. The ISO 31700 standard provides high-level advice and examples, but lacks specific technical guidance. While these principles help define design requirements, they do not address how to bridge the gap between general laws and building systems that respect user privacy. Our work aims to bridge this gap.

We have developed privacy-by-design methods and tools that are (i) *rigorous*, supporting precise formalizations of legal requirements, (ii) *mechanizable*, providing runtime policy enforcement and code-generation, and (iii) *general*, applying to diverse privacy requirements and implementations. We build on the notion of runtime enforcement (RE) [24], where a policy enforcement point (PEP) intercepts attempted system actions, converts them to events, and sends them to a policy decision point (PDP), as in Figure 1. In response to events or proactively as time passes, the PDP issues commands to ensure policy compliance at all times; based on these commands, the PEP forces the system to take modified, compliant actions.

We present highlights of two approaches to achieving this, both building on RE in different ways. The first approach (Section 2), embodied in the EnfGuard [18, 19] and InstrLib [13] tools, proposes a novel PDP algorithm for expressive specifications written in metric first-order temporal logic (MFOTL) [15] and a general PEP library for Python applications. MFOTL's expressiveness and its ability to proactively send commands makes EnfGuard a powerful tool that can be used with any enforceable formalization of privacy requirements. The InstrLib library interfaces the system with EnfGuard by intercepting specified actions and replacing them with those actions prescribed by EnfGuard's commands.

The second approach (Section 3), implemented in the *v*ActionGUI tool [20], is a substantial extension of the idea of Model-Driven Security [2, 22] to privacy. Namely, systems are designed with formal data and privacy models. The privacy models formulate data-protection requirements like purpose-based usage and user consent. Model transformations generate correct-by-design code for the enforcement mechanisms directly from the models themselves.

We provide details on both approaches, including tool support and their evaluations. We conclude by comparing these approaches and pointing to directions for future work (Section 4).

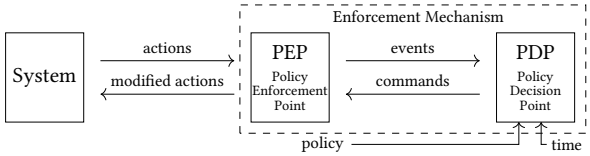

Figure 1: Runtime enforcement model

*Keynote speaker

CCS '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3748271

Syntax	Semantics
$\mathsf{E}(a_1, \ldots, a_n)$	The event $\mathsf{E}(a_1, \ldots, a_n)$ occurs in the present
$\phi \wedge \psi$; $\phi \rightarrow \psi$	ϕ AND ψ hold; ϕ holds IMPLIES ψ holds
$\neg\phi$; $\forall u.\ \phi$	ϕ does NOT hold; FOR ALL values of x, ϕ holds
$\Diamond\,\phi$; $\Box\,\phi$	EVENTUALLY, ϕ will hold; ϕ ALWAYS holds
$\psi\ \mathsf{S}\ \phi$	Some time in the past, ϕ held and SINCE then, ψ has always held

Each temporal operator ($\Diamond$,$\Box$, S) can be subscripted with an interval $I = [a, b]$ or $I = [a, \infty)$ that specifies a time window in which ϕ must hold. For instance, $\Box_{[0,60]}\,\phi$ means 'ϕ ALWAYS holds during the next 60 time units.'

Figure 2: MFOTL syntax and semantics (selected)

2 Proactive Runtime Enforcement

Logical formalisms have been previously proposed to formalize privacy laws [9, 12, 23, 25]. Applying these approaches to runtime policy *enforcement*, rather than just *detecting* non-compliance (as in runtime monitoring [11]), poses new challenges.

The first challenge is expressivity: the formalism should be able to specify a wide range of requirements and assumptions about the system. This includes explicitly specifying the temporal sequencing of events, which has been the source of errors in previous work [17]. In the context of runtime enforcement, the formalization should capture not only obligations, but also assumptions on how the PEP can modify events. For instance, if data processing requires consent, then, logically, one way to ensure compliance is to force the logging of consent whenever processing happens! This, however, would defeat the principle of free consent. Hence, the PEP should be constrained to *never cause consent* events, but to be able to *suppress processing* events whenever needed.

The second challenge is performance: the PDP and PEP used should operate in real time with little runtime overhead, also when composed with other software modules to enforce policies in complex applications.

For high expressivity, we opted for metric first-order temporal logic (MFOTL) [6] as a policy language and developed several efficient PDP tools for MFOTL. We then used our tools to enforce privacy requirements in web applications at runtime, demonstrating our approach's practical performance.

Formalizing and enforcing MFOTL specifications

In earlier work on monitoring GDPR requirements using industry log files, Arfelt et al. [1] showed that MFOTL could be used to formalize core GDPR requirements. In addition to standard propositional connectives, MFOTL (Figure 2) features temporal operators with intervals specifying real-time constraints, events with arguments taken from an infinite domain (e.g., integers or strings), and quantifiers. MFOTL has been recently extended to support additionally SQL-style aggregations [5], (recursive) let bindings [26], and function applications in terms [18]. This logic is at the core of a mature ecosystem of tools and case studies that demonstrate its relevance to real-world applications [7].

Example 2.1. Consider the following requirements adapted from the GDPR: (R1) when a user's personal data is processed for some purpose, the user should have consented to the processing and not revoked this consent; (R2) whenever a user requests the deletion of their personal data, then this data should be deleted within 30 days.

In the context of runtime enforcement, such requirements can be formalized by: a *signature*, consisting of a list of relevant events; a set of *capabilities* for each event, indicating whether the event can be caused or suppressed by the PEP; and a *formula* (also called 'policy') describing compliant sequences of events.

For (R1), we introduce four events, $\mathsf{Use}(d, p)$, $\mathsf{PersonalData}(d, u)$, $\mathsf{GiveConsent}(u, p)$, and $\mathsf{RevokeConsent}(u, p)$, that read as: 'data d is used/processed for purpose p', 'data d is personal data relating to user u', 'user u consents to the processing of their data for purpose p', and 'user u revokes their consent to use their data for purpose p'. The event Use is suppressable and the other three events are only observable. The MFOTL formula encoding (R1) is

$$\Box(\forall u, d, p.\ \mathsf{Use}(d, p) \wedge \mathsf{PersonalData}(d, u) \rightarrow (\neg\mathsf{RevokeConsent}(u, p)\ \mathsf{S}\ \mathsf{GiveConsent}(u, p))). \quad (1)$$

To formalize (R2), we use two additional events, $\mathsf{DeleteReq}(u)$ and $\mathsf{Delete}(d)$, meaning 'user u requests deletion of their personal data' and 'data d is deleted', respectively. To enforce this requirement, Delete should be causable by the PEP, whereas DeleteReq needs only be observable. Assuming that the system's time unit is seconds, the corresponding formula is

$$\Box(\forall u.\ \mathsf{DeleteReq}(u) \rightarrow \Diamond_{[0,2592000]}\ \forall d.\ (\mathsf{PersonalData}(d, u) \rightarrow \mathsf{Delete}(d))), \quad (2)$$

where 2592000 is the number of seconds in 30 days.

In recent work, we implemented three PDPs for MFOTL: EnfPoly [14], WhyEnf [19], and EnfGuard [18], which support increasingly large fragments of MFOTL. WhyEnf and EnfGuard are *proactive* PDPs: they not only respond to user events, but can also insert events before a deadline ('in the nick of time' [3]), to discharge obligations. This feature is important when enforcing policies such as (2), which require some action—here, Delete—to be taken independently of system events. EnfGuard additionally supports function applications, aggregations, and let bindings, making it the most expressive MFOTL enforcer available. Each of these tools takes as input a signature, a set of capabilities, and a formula. It first statically checks whether the provided formula is enforceable according to the given capabilities, and, when this is the case, proceeds to enforce the formula at runtime. Since first-order enforceability is undecidable [14], we use typing rules to identify an enforceable syntactic fragment of MFOTL [18, 19]. We demonstrate that our enforceable fragment is expressive enough to support a large set of both privacy and non-privacy benchmarks found in previous work [18].

Runtime enforcement in web applications

To evaluate our tools' practicality, we conducted multiple case studies [13, 15, 16] that use our PDPs as backends for enforcing privacy requirements in web applications at runtime.

First, we built a prototype web development framework that supports the deployment of untrusted applications processing user data [16]. Users can define custom privacy requirements in MFOTL. The PEP uses dynamic information-flow control to track the flows of information in the code and produce appropriate events that are

checked by EnfPoly for compliance with the user-defined requirements. We ported three existing applications to our framework with modest latency overhead (≤15 ms).

We subsequently extended these case studies to enforce six key GDPR requirements in web applications [15]. In our extended framework, called WebTTC+, a privacy dashboard is available for users to exercise their rights, e.g., request the deletion of their data. Additionally, users can use a browser extension to give or withdraw consent for any of their inputs to an application. The latency overhead remains moderate (≤10 ms).

More recently, we developed a lightweight instrumentation library for EnfGuard, which supports the enforcement of arbitrary Python code with EnfGuard with minimal coding effort [13]. This new library and PDP are at the core of an ongoing effort to build larger case studies of web privacy enforcement.

3 Model-driven Privacy

For developers, ensuring that systems comply with privacy requirements is a difficult, error-prone task. To address this challenge, we propose a model-driven system-development methodology, focusing on purpose-based data usage and user consent requirements [20]. As indicated in Table 1, developers define data and privacy models, from which our tools automatically generate an application with a fully-configured privacy enforcement mechanism.

Developer defines	Our approach generates
Data model	Data classes & method stubs
Privacy model	Runtime instrumentation
– user class	Authentication
– personal data classes	Privacy classes & helper methods
– purposes and their hierarchy	Database configuration
– declared purposes	Privacy notice & consent management
– actual purpose mapping	Purpose tracking mechanism

Table 1: Model-driven privacy methodology

Modeling and enforcing privacy requirements

In our approach, developers first specify a *data model*, describing the system's state space and privacy-relevant aspects such as what constitutes personal data. The model, given by a UML class diagram, describes a set of classes and enumeration types, each with their own attributes and methods. Classes may be related by associations.

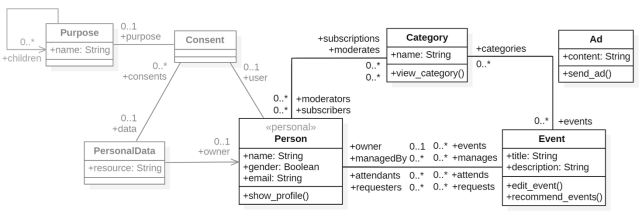

Figure 3: Data model of an event management platform

Example 3.1. Figure 3 (black part) shows a data model of an event management platform that includes the classes `Person`, `Event`, `Category`, and `Ad`, related by seven associations. Each class defines a set of attributes and methods; for example, the `Person` class includes an `email` attribute and a `show_profile()` method.

Based on the data model, the *privacy model* identifies the *user class* and classes containing *personal data*. It further defines a set of *purposes* and an associated *purpose hierarchy*. It also contains a *privacy policy*, which is a set of *declared purposes*, each specifying: (1) personal data, (2) the purpose of its use, and (3) a condition under which usage is permitted. Finally, the privacy model defines a *purpose mapping*, relating some methods to their actual purposes.

Example 3.2. Consider a privacy policy of the event platform:

- We use your subscription information to recommend you events, but only if you have attended fewer than three.
- We use your email address information for functional purposes, such as displaying your profile.

The privacy model includes Profile, Recommendations, and the composite Functional purpose, which encompasses Profile. The `Person` class is both the user and personal data class. Besides the above policy, the model maps `recommend_events()` and `show_profile()` to Recommendations and Profile, respectively.

Prior to generating the PDP and PEP, our approach (and associated tools) first performs a model-to-model transformation, integrating parts of the privacy model into the data model, i.e., introducing a `Consent` class that links the data owner, their personal data, and a purpose (Figure 3, gray part). It afterward generates the corresponding classes with method stubs, authentication, database configurations, and a privacy enforcement mechanism. This mechanism includes: a privacy notice where users can grant or revoke consent for each declared purpose; a purpose-tracking mechanism that maintains a stack of active purposes during execution by pushing the purpose of each called method (as defined in the purpose mapping) and popping it on return; and runtime instrumentation that intercepts personal data actions and ensures they are allowed only if every purpose on the stack matches a declared purpose whose condition holds and for which the data owner has given consent.

Finally, it is the developer's responsibility to implement the application's functional requirements by writing code for the generated method stubs. Depending on this implementation, different data actions, such as reading (e.g., `v = user.email`) or updating personal data (e.g., `user.email = v`), may be triggered at runtime. The system's instrumentation intercepts these actions during execution. If a privacy violation were to occur, the read action would return a restricted default value, whereas other actions would raise a privacy exception. Through this instrumentation of data actions, our approach ensures that privacy policies are consistently enforced throughout the system (complete mediation). This helps developers avoid accidental privacy violations and thereby reduces the risk of non-compliance.

Example 3.3. Consider the following Python code snippet, in which the developer replaces the generated method stub of the `show_profile()` method with a concrete implementation:

```
# Generated method stub:
def show_profile():
    # Code implemented by developers
    ucategories = current_user.subscriptions # a violation
    uemail = current_user.email              # not a violation
    return render_template('profile.html',{'categories': ucategories,
                                           'email': uemail})
```

When the user *u* executes this method, the system pushes the Profile purpose onto the active purpose stack. Based on Example 3.2,

Approach	Capabilities	Activeness	Expressiveness	Policy Language	Integration	References
Proactive Enforcement	Suppression & Causation	Proactive	Enforceable GDPR	MFOTL	general PDP & manual PEP	[13, 15, 18, 19]
Model-Driven Privacy	Suppression only	Reactive	Purpose limitation	OCL	generated PDP & PEP	[4, 20, 21]

Table 2: Comparison of our two approaches

reading u's `subscriptions` violates the privacy policy and is always suppressed. In contrast, reading u's email is permitted only if u has given consent for either Functional or Profile purposes.

Evaluation

To demonstrate the practicality of this approach, we developed model-to-code transformations for two popular web frameworks: ASP.NET (C#) and Flask (Python). The latter transformation, called *v*ActionGUI, is publicly available at [21]. Our performance evaluation shows that the instrumentation on data actions introduces modest overhead, e.g., viewing a personal timeline on a social networking app incurs a fraction of a second per request.

We also conducted an empirical study in a graduate-level security engineering course. Sixty students implemented privacy requirements for the event management platform (Figure 3) twice: once manually and once using the *v*ActionGUI tool. Despite having less experience with model-driven development and tools, students using *v*ActionGUI required 33% less time and wrote eight times less code to implement applications that are, on average, 64% less prone to privacy violations.

4 Conclusions

We addressed the challenge of providing technical foundations for privacy by design using two runtime enforcement approaches. The first one, based on the EnfGuard and InstrLib tools, applies metric first-order temporal logic to specify and proactively enforce privacy requirements. The second one, based on the *v*ActionGUI tool, uses model-driven development to generate applications with built-in privacy enforcement. Both have been evaluated in practical case studies, showing their effectiveness and low runtime overhead.

The two approaches complement each other. The former supports a more expressive policy language and can *proactively* enforce a wider range of requirements. In contrast, the latter focuses on the most critical aspect of privacy. Moreover, it is more developer-friendly and simpler to integrate into existing development workflows as it generates the policy enforcement point automatically, without requiring a manual instrumentation effort (Table 2).

Still, achieving rigorous, tool-supported privacy by design faces further challenges. Supporting collaboration between lawyers and system designers to audit IT systems for legal compliance is essential, especially in linking laws to system actions. Additionally, enforcing data protection aspects like data minimization [10, Art. 5(1)(c)], which cannot be enforced by observing individual executions, is an exciting topic to explore.

Acknowledgments

François Hublet and Hoàng Nguyễn are supported by SNSF grant 204796, "Model-driven Security & Privacy".

References

[1] Emma Arfelt, David Basin, and Søren Debois. 2019. Monitoring the GDPR. In *European Symposium on Research in Computer Security (ESORICS)*, Vol. 11735. Springer, 681–699.
[2] David Basin, Manuel Clavel, and Marina Egea. 2011. A decade of model-driven security. In *Access control models and technologies (SACMAT)*, Vol. 1998443. 1–10.
[3] David Basin, Søren Debois, and Thomas Hildebrandt. 2024. Proactive enforcement of provisions and obligations. *Journal of Computer Security* 32, 3 (2024), 247–289.
[4] David Basin, Juan Guarnizo, Srđan Krstić, Hoang Nguyen, and Martín Ochoa. 2023. Is Modeling Access Control Worth It?. In *Computer and Communications Security (CCS)*. 2830–2844.
[5] David Basin, Felix Klaedtke, Srdjan Marinovic, and Eugen Zălinescu. 2015. Monitoring of temporal first-order properties with aggregations. *Formal methods in system design* 46 (2015), 262–285.
[6] David Basin, Felix Klaedtke, Samuel Müller, and Eugen Zălinescu. 2015. Monitoring Metric First-Order Temporal Properties. *J. ACM* 62, 2 (May 2015), 15:1–15:45.
[7] David Basin, Srđan Krstić, Joshua Schneider, and Dmitriy Traytel. 2023. Correct and efficient policy monitoring, a retrospective. In *Automated Technology for Verification and Analysis (ATVA)*, Vol. 14215. Springer, 3–30.
[8] Ann Cavoukian. 2011. *Privacy by Design: The 7 Foundational Principles – Implementation and Mapping of Fair Information Practices.* Technical Report.
[9] Henry DeYoung, Deepak Garg, Limin Jia, Dilsun Kaynar, and Anupam Datta. 2010. Experiences in the logical specification of the HIPAA and GLBA privacy laws. In *Workshop on Privacy in the Electronic Society*. 73–82.
[10] European Parliament and Council of the European Union. 2016. General Data Protection Regulation, (EU) 2016/679. https://data.europa.eu/eli/reg/2016/679/oj
[11] Yliès Falcone, Srđan Krstić, Giles Reger, and Dmitriy Traytel. 2021. A taxonomy for classifying runtime verification tools. *Int. J. Softw. Tools Technol. Transf.* 23, 2 (2021), 255–284.
[12] Emil Heuck, Thomas Hildebrandt, Rasmus Kiærulff Lerche, Morten Marquard, Håkon Normann, Rasmus Iven Strømsted, and Barbara Weber. 2017. Digitalising the general data protection regulation with dynamic condition response graphs. In *Business Process Management*. 124–134.
[13] François Hublet, David Basin, Linda Hu, Srđan Krstić, and Lennard Reese. 2025. Instrumenting Runtime Enforcement. In *Runtime Verification (RV)*. Springer.
[14] François Hublet, David Basin, and Srđan Krstić. 2022. Real-Time Policy Enforcement with Metric First-Order Temporal Logic. In *European Symposium on Research in Computer Security (ESORICS)*, Vol. 13555. 211–232.
[15] François Hublet, David Basin, and Srđan Krstić. 2023. Enforcing the GDPR. In *European Symposium on Research in Computer Security*, Vol. 14345. 400–422.
[16] François Hublet, David Basin, and Srđan Krstić. 2024. User-Controlled Privacy: Taint, Track, and Control. *Privacy Enhancing Technologies* 2024, 1 (2024), 597–616.
[17] François Hublet, Alexander Kvamme, and Srđan Krstić. 2024. Towards an Enforceable GDPR Specification. In *Mapping and Governing the Online World (MGOW)*.
[18] François Hublet, Leonardo Lima, David Basin, Srđan Krstić, and Dmitriy Traytel. 2025. Proactive Real-Time First-Order Enforcement. In *Computer Aided Verification (CAV)*, Vol. 15933. Springer, 370–392.
[19] François Hublet, Leonardo Lima, David Basin, Srđan Krstić, and Dmitriy Traytel. 2024. Proactive Real-Time First-Order Enforcement. In *Computer Aided Verification (CAV)*, Vol. 14682. Springer, 156–181.
[20] Srđan Krstić, Hoang Nguyen, and David Basin. 2024. Model-driven Privacy. *Privacy Enhancing Technologies* 2024, 1 (2024), 314–329.
[21] Srđan Krstić, Hoang Nguyen, and David Basin. 2025. The *v*ActionGUI tool. https://nuactiongui.github.io/.
[22] Torsten Lodderstedt, David Basin, and Jürgen Doser. 2002. SecureUML: A UML-Based Modeling Language for Model-Driven Security. In *5th International Conference on The Unified Modeling Language (UML) (LNCS, Vol. 2460)*. Springer, 426–441. doi:10.1007/3-540-45800-X_33
[23] Livio Robaldo, Cesare Bartolini, Monica Palmirani, Arianna Rossi, Michele Martoni, and Gabriele Lenzini. 2020. Formalizing GDPR provisions in reified I/O logic: the DAPRECO knowledge base. *Journal of Logic, Language and Information* 29 (2020), 401–449.
[24] Fred Schneider. 2000. Enforceable security policies. *Transactions on Information and System Security* 3, 1 (2000), 30–50.
[25] L. Thomas Van Binsbergen, Lu-Chi Liu, Robert Van Doesburg, and Tom Van Engers. 2020. eFLINT: a domain-specific language for executable norm specifications. In *Conference on Generative Programming: Concepts and Experiences*. 124–136.
[26] Sheila Zingg, Srđan Krstić, Martin Raszyk, Joshua Schneider, and Dmitriy Traytel. 2022. Verified first-order monitoring with recursive rules. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Vol.13244.Springer,236–253.