# Formal Modeling and Analysis of Data Protection for GDPR Compliance of IoT Healthcare Systems

Florian Kammüller
Middlesex University London, UK
f.kammueller@mdx.ac.uk

***Abstract*—In this paper, we investigate the implications of the General Data Privacy Regulation (GDPR) on the design of an IoT healthcare system. From 26th May 2018, the GDPR will become mandatory within the European Union and hence also for any supplier of IT products. Breaches of the regulation will be fined with penalties of 20 Million EUR. This is a strong motivation for system designers to enable the proof of compliance to the GDPR. We propose the use of formal modeling and analysis using interactive theorem proving. Based on previous work on modeling infrastructures and security policies for insider attacks, we demonstrate the use of logical modeling and machine assisted verification to support data protection (privacy) by design. We illustrate this process on the case study of IoT based monitoring of Alzheimer's patients that we work on in the CHIST-ERA project SUCCESS.**

## I. Introduction

Infringements on the basic principles of data processing, rights of data subjects, or other non-compliances to Articles of the GDPR are fined with up to 20 million EUR or 4% of the annual turnover of an undertaking whichever is higher (Article 79 (3a) [1]). Therefore, it is a crucial need for any company to find ways to achieve and keep GDPR compliance. The General Data Privacy Regulation (GDPR) [1] is a 209 page legal document. For small businesses, it might be hard to tackle such a complex requirements specification. For these reasons, we strife in this paper, to reduce some of that complexity and overcome the difficulty of legal formulation by

(a) summarizing the legal text, highlighting the technically relevant parts and
(b) providing an abstract model of infrastructures with actors and policies in which the requirements given by the GDPR are formally specified in logic.

We thus provide an important first step towards producing GDPR compliant systems by establishing a general framework for creating a formal system design that provably complies to the regulation. The latter strong guarantee is achieved because the framework we suggest is based on the interactive theorem prover Isabelle. Building on previous work of modeling and verification of Insider threats using logical analysis [2], we generalize and extend that framework for general privacy and security analysis of *infrastructures*. In particular, we add a completely new layer of data labeled with security labels in the style of the Decentralized Label Model (DLM) [3].

In our approach we complement the convenience of temporal logics and Kripke structures with the expressive power of Higher Order Logic (HOL) of Isabelle. This enables

(1) the formalisation of precise yet comprehensible system design specifications. These specifications can then be used to formally
(2) express data protection (privacy) properties of the GDPR and prove them in the Isabelle framework using machine assistance.

The combination of Steps (1) and (2) provides formally verified data protection (privacy) by design.

Addressing point (a), we first give an overview of the regulation pointing to the parts relevant for technical design of information systems (Section II). We then address point (b) by presenting our modeling and analysis technique in Isabelle (Section III) showing how GDPR relevant properties can be formalized using data types and state transition rules (Section IV). In Section V, we illustrate the application of the framework as a case study of our CHIST-ERA project SUCCESS [4] on security and privacy of IoT healthcare systems showing how temporal logic can be used to express GDPR compliance naturally in Isabelle. We conclude in Section VI.

## II. The European Standard GDPR

The GDPR (General Data Protection Regulation) is in full called "the regulation of the European parliament and the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data". For this paper, we use the final proposal [1] as our source to provide a comprehensive summary of the main points relevant for a technical analysis. Despite the relatively large size of the document of 209 pages, the relevant portion for this is only about 30 pages (Pages 81–111, Chapters I to Chapter III, Section 3). In more detail, the GDPR is constituted by the following parts (according to page numbers):

- 1–5: Introduction
- 6–80: The ANNEX starts with a listing of sections numbered (1)...(135) addressing various aspects of the regulation with no obvious subject structure but containing valuable information clarifying and premeditating important concepts, ideas and concepts. It might be considered as prologue to the actual regulation text.
- 81–209: Chapters I–XI contain 91 articles with subsections (paragraphs) ordering them along major purposes and themes. More complex chapters are divided into sections.

  In more detail these chapters are:

Chapter I, 81–87, *General Provisions*, Articles (1–4):
Subject matter and objectives, Material scope, Territorial scope, Definitions.

Chapter II, 88–96, *Principles*, Articles (5–10):
Principle relating to personal data processing, Lawfulness of processing, Conditions for consent, Processing of data.

Chapter III, 96–115, *Rights of the Data Subject*, Articles (11–21):
Section 1 (Transparency and modalities), Section 2 (Information and access to data), Section 3 (Rectification and erasure), Section 4 (Right to object and automated individual decision making), Section 5 (Restrictions).

Chapter IV, 116–144, *Controller and Processor*, Articles (22–39a):
Section 1 (General obligations), Section 2 (Data Security), Section 3 (Data protection assessment and prior consultation), Section 4 (Data protection officer), Section 5 (Codes of conduct and certification).

Chapter V, 145–156, *Transfer of Personal Data to Third Countries or International Organisations*, Articles (40–45).

Chapter VI, 157–168, *Independent Supervisory Authorities*, Articles (46–54):
Section 1 (Independent status), Section 2 (Competence, Tasks and Powers).

Chapter VII, 169–181, *Co-operation and Consistency*, Articles (54a–72):
Section 1 (Co-operation), Section 2 (Consistency), Section 3 (European Data Protection Board).

Chapter VIII, 182–199, Remedies, Liability, and Sanctions, Articles (73–79b).

Chapter IX, 200–204, Provisions Relating to Specific Data Processing Situations, Articles (80–85).

Chapter X, 205–206, *Delegated Acts and Implementing Acts*, Articles (86–87).

Chapter XI, 207–209, *Final Provisions*, Articles (88–91).

The central laws for enforcing Security and Privacy are Articles 23 *Data protection by design and by default* and Article 30 *Secure Processing*. However, the technically relevant parts providing the detailed definition of terms and functional compliance requirements of systems are contained in the earlier Articles (1–21) in Chapter I to Chapter III, Section 3.

Chapter I generally defines the scope of the regulation in terms of main purpose (protection of individuals), material scope (personal data) and territories (in the Union); Article 4 provides definitions (1–21) for main terms like *personal data*, *processing*, *profiling*, *controller*, or *third party* amongst others. Thus Chapter I is relevant for the basic definitions of our model in Section III.

Chapter II defines the principles for data processing and retention. Article 5 states that data must be processed "lawfully, fairly and in a transparent manner" [1] and furthermore the article defines *purpose limitation, data minimisation, accuracy, storage limitation, integrity and confidentiality*, and *accountability*. Article 6 then details the notion of *lawfulness*. Article 7 specifies the *conditions for consent* which is requested for any processing of data concerning the data subject. Article 8 adds the details for a child's consent. Article 9 is dedicated to the processing of special categories of personal data, for example, revealing ethnic origin, political opinions or religious or philosophical beliefs and provides in Sentence 2 a list of exceptions where data protection is exempt. Article 9a defines further exemptions in case of criminal convictions and offences. Article 10 is very relevant for the use of data for scientific purposes: *Processing not requiring identification*. Sentence 1 relieves the duty of data identification if the purpose of processing does no longer require this. Sentence 2 specifies the exemption from the central Articles 15–18 on access and erasure rights of the data subject that we explain next.

While Chapters I and II provide essential definitions, more technical requirements for data processing are provided in Chapter III, Sections 1 to 3.

- Section 1 describes Transparency and Modalities. Article 12 states that the controller must provide any information and communication (specified in Article 14–20) "relating to the data subject in a concise transparent and intelligible and easily accessible form ...".
- Section 2 provides details of the access rights and the information that the controller must provide to a data subject on request (Articles 14, 14a, 15), like the retention time and the purpose of data collection.
- Section 3 defines the right of a data subject to rectification and erasure of personal data ("right to be forgotten") as well as the right to restrict its processing (Articles 15–18).

In summary, Chapter III specifies that the controller must give the data subject *read access* (1) to any information, communications, and "meta-data" of the data, e.g., retention time and purpose. In addition, the system must enable *deletion of data* (2) and restriction of processing.

An invariant condition for data processing resulting from these Articles is that the system *functions* must *preserve* any of the access rights of personal data (3).

## III. Formal Model in Isabelle

In this section, we describe the Isabelle framework for security and privacy analysis of infrastructures with policies.

Isabelle is a generic Higher Order Logic (HOL) proof assistant. Its generic aspect allows the embedding of so-called object-logics as new theories on top of HOL. There are sophisticated proof tactics available to support reasoning: simplification, first-order resolution, and special macros to support arithmetic amongst others. The use of HOL has the advantage that it enables expressing even the most complex application scenarios, conditions, and logical requirements. Isabelle enables the analysis of meta-theory, that is, we can prove theorems *in* an object logic but also *about* it.

Object-logics are added to Isabelle using constant and type definitions forming a so-called *conservative extension*. That is, no inconsistency can be introduced: new types are defined as

subsets of existing types; properties are proved using a one-to-one relationship to the new type from properties of the existing type.

### A. Isabelle Framework for Infrastructure Security and Privacy

This framework has been created initially for the modeling and analysis of Insider threats [2]. Its use has been validated on the most well-known insider threat patterns identified by the CERT-Guide to Insider threats [5]. More recently, this Isabelle framework has been successfully applied to realistic case studies of insider attacks in airplane safety [6] and on auction protocols [7]. These larger case studies as well as complementary work on the analysis of Insider attacks on IoT infrastructures [8]–[10] have motivated the extension of the original framework by Kripke structures and temporal logic [11] as well as a formalisation of attack trees [12].

In the course of this extension, the Isabelle framework has been restructured such that it is now a general framework for the state-based security analysis of infrastructures with policies and actors. Temporal logic and Kripke structure build the foundation. Meta-theoretical results have been established to show equivalence between attack trees and CTL statements. This foundation provides a generic notion of state transition on which attack trees and temporal logic can be used to express properties. Interactive proof is used to prove these properties but the meta-theory can be applied to immediately produce results. Figure 1 gives an overview of the framework.

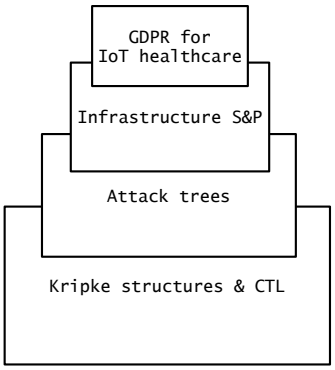


Fig. 1. Generic framework for infrastructrures embeds applications.

Without going into too much detail, we will provide a short introduction to provide just enough detail to understand how we model the GDPR requirements.

### B. Kripke Structure, CTL, and Attack Trees

We apply Kripke structures and CTL to model state based systems and analyse properties under dynamic state changes. Snapshots of systems are the states on which we define a state transition. Temporal logic is then employed to express security and privacy properties.

In Isabelle, the system states and their transition relation are defined as a *type class* called `state` containing an abstract constant `state_transition`. It introduces the syntactic infix notation `I` $\rightarrow_i$ `I'` to denote that system state `I` and `I'` are in this relation.

```
class state =
fixes state_transition :: [σ :: type, σ] ⇒ bool
                                          ("_  →ᵢ _")
```

The above class definition allows lifting Kripke structures and CTL to a general level. Branching time temporal logic CTL is defined over Kripke structures with such state transitions and can be applied to the theory of infrastructures.

Based on the generic state transition $\rightarrow_i$ of the `state` type class, the CTL-operators EX and AX express that property $f$ holds in some or all next states, respectively. The CTL formula AG $f$ means that on all paths branching from a state $s$ the formula $f$ is always true (G stands for 'globally'). It can be defined using the Tarski fixpoint theory by applying the greatest fixpoint operator. In a similar way, the other CTL operators are defined. The formal Isabelle definition of what it means that formula $f$ holds in a Kripke structure `M` can be stated as: the initial states of the Kripke structure `init M` need to be contained in the set of all states `states M` that imply $f$.

```
M ⊢ f ≡  init M ⊆ { s ∈ states M. s ∈ f  }
```

In an application, the set of states of the Kripke structure will be defined as the set of states reachable by the infrastructure state transition from some initial state, say `example_scenario`.

```
example_states ≡ { I. example_scenario →ᵢ^*  I }
```

The relation $\rightarrow_i$`^*` is an operator supplied by the Isabelle theory library. It is the reflexive transitive closure applied to the relation $\rightarrow_i$.

The `Kripke` constructor combines the constituents initial state, state set and state transition relation $\rightarrow_i$.

```
example_Kripke ≡
    Kripke example_states {example_scenario} →ᵢ
```

In Isabelle, the concept of sets and predicates coincide[1]. Thus a `property` is a predicate over states which is equal to a set of states. For example, we can then try to prove that there is a path (E) to a state in which the property eventually holds (in the Future) by starting the following proof in Isabelle.

```
example_Kripke ⊢ EF property
```

### C. Infrastructures, Policies, and Actors

The Isabelle Infrastructure framework supports the representation of infrastructures as graphs with actors and policies attached to nodes. These infrastructures are the *states* of the Kripke structure.

The transition between states is triggered by non-parametrized actions `get`, `move`, `eval`, and `put` executed by actors. Actors are given by an abstract type `actor` and a function `Actor` that creates elements of that type from

[1]In general, this is often referred to as *predicate transformer semantics.*

identities (of type string). Policies are given by pairs of predicates (conditions) and sets of (enabled) actions.

```
type_synonym policy = ((actor ⇒ bool) × action set)
```

Actors are contained in an infrastructure graph.

```
datatype igraph =
        Lgraph (location × location)set
               location ⇒ identity set
               actor ⇒ (string set × string set)
               location ⇒ (string × acond)
```

An igraph contains a set of location pairs representing the topology of the infrastructure as a graph of nodes and a list of actor identities associated to each node (location) in the graph. Also an igraph associates actors to a pair of string sets by a pair-valued function whose first range component is a set describing the credentials in the possession of an actor and the second component is a set defining the roles the actor can take on. More importantly in this context, an igraph assigns locations to a pair of a string that defines the state of the component and an element of type acond. This type acond is defined as a set of labelled data representing a condition on that data. Corresponding projection functions for each of these components of an igraph are provided; they are named gra for the actual set of pairs of locations, agra for the actor map, cgra for the credentials, and lgra for the state of a location and the data at that location.

Infrastructures are given by the following datatype that contains an infrastructure graph of type igraph and a policy given by a function that assigns local policies over a graph to all locations of the graph.

```
datatype infrastructure =
        Infrastructure
            igraph
            [igraph, location] ⇒ policy set
```

There are projection functions graphI and delta when applied to an infrastructure return the graph and the policy, respectively. Policies specify the expected behaviour of actors of an infrastructure. They are defined by the enables predicate: an actor h is enabled to perform an action a in infrastructure I, at location l if there exists a pair (p,e) in the local policy of l (delta I l projects to the local policy) such that the action a is a member of the action set e and the policy predicate p holds for actor h.

```
enables I l h a ≡ ∃ (p,e) ∈ delta I l. a ∈ e ∧ p h
```

Equipped with the basic Infrastructure constructors, we consider next how these can be used to specify a GDPR compliant system.

## IV. Data Protection by Design for GDPR compliance

### A. Security and Privacy by Labeling Data

The Decentralised Label Model (DLM) [3] introduced the idea to label data by owners and readers. We pick up this idea and formalize a new type to encode the owner and the set of readers.

```
type_synonym dlm = actor × actor set
```

Labelled data is then just given by the type dlm × data where data can be any data type. Additional meta-data, like retention time and purpose, can be encoded as part of this type data. We omit these detail here for conciseness of the exposition.

Using labeled data, we can now express the essence of Article 4 Paragraph (1): 'personal data' means any information relating to an identified or identifiable natural person ('data subject'). Since we have a more constructive system view, we express this by defining the owner of a data item d of type dlm as the actor that is the first element in the pair that is the first of the pair d. Then, we use this function to express the predicate "owns".

```
definition owner :: dlm × data ⇒ actor
where owner d ≡ fst(fst d)

definition owns ::
  [igraph, location, actor, dlm × data] ⇒ bool
where owns G l a d ≡ owner d = a
```

The introduction of a similar function for readers projecting the second element of a dlm label

```
definition readers :: dlm × data ⟹ actor set
where readers d ≡ snd (fst d)
```

enables specifying when an actor may access a data item.

```
definition has_access ::
  [igraph, location, actor, dlm × data] ⇒ bool
where has_access G l a d ≡
           owns G l a d ∨ a ∈ readers d
```

### B. Privacy Preserving Functions

The labels of data must not be changed by processing: we have identified this finally as an invariant (3) resulting from the GDPR in Section II. This invariant can be formalized in our Isabelle model by a type definition of functions on labeled data that preserve their labels.

```
typedef label_fun = {f :: dlm × data ⟹ dlm × data.
                     ∀ x. fst x = fst (f x)}
```

We also define an additional function application operator ⇳ on this new type. Then we can use this restricted function type to implicitly specify that only functions preserving labels may be applied in the definition of the system behaviour in the following state transition rules.

### C. Data Protecting State Transition

The abstract state transition provided in the underlying Kripke structure theory is instantiated in the GDPR infrastructure model by an inductive definition of a state transition relation $\rightarrow_n$ over infrastructures. A set of inductive rules defines this transition relation $\rightarrow_n$ relative to characteristics of the current state. These characteristics can exploit the information encoded into the infrastructure as well as the enables predicate to express how the next infrastructure state evolves from the current one. It is here where the GDPR

articles are incarnated into the system specification. We show here the rules for put, get, process and delete (see the complete source code [13] for full details).

*1) The put pata rule:* assumes an actor `a` residing at a location `l` in the infrastructure graph `G` and being enabled the `put` action. If infrastructure state `I` fulfils those preconditions, the next state `I'` can be constructed from the current state by adding the data item `((Actor a, as), n)` at location `l`. The addition is given by updating (using `:=`) the existing data storage `lgra G l` at location `l` with the set union of its second element and the singleton set $\{$`((Actor a, as), n)`$\}$. Note that the first component `Actor a` marks the owner of this data item as `a`. The other components, the reader list `as`, and the actual data `n`, as well as the state of the location `s` can be instantiated freely within the limitations given by the Isabelle types.

```
put:
G = graphI I ⟹ a @_G l ⟹
enables I l (Actor a) put ⟹
I' = Infrastructure
     (Lgraph (gra G)(agra G)(cgra G)
       ((lgra G)(l :=
        (s, snd (lgra G l) ∪ {((Actor a, as), n)}))))
      (delta I)
⟹ I →_n I'
```

*2) The get data rule:* resembles the put data rule in many parts. However, here an actor `a` accesses data in a remote location `l'` and adds it to the data in his current location `l`. This copying of data is only permitted if the current location `l'` of the data enables `a` to `get` and if the list of readers `as` in the data item `((Actor a', as), n)` contains the entry `Actor a`. Different to the put rule, this rule preserves the first component `fst(lgra G l)` of the state of location `l`.

```
get_data:
G = graphI I ⟹ a @_G l ⟹
enables I l' (Actor a) get ⟹
((Actor a', as), n) ∈ snd (lgra G l') ⟹
Actor a ∈ as ⟹
I' = Infrastructure
     (Lgraph (gra G)(agra G)(cgra G)
       ((lgra G)(l := (fst (lgra G l),
          snd (lgra G l) ∪ {((Actor a', as), n)}))))
      (delta I)
⟹ I →_n I'
```

*3) The process rule:* prescribes how data within the infrastructure may be processed. It imposes that only privacy preserving functions may be applied to data (see Section IV-B). This is achieved by using the application operator ⇳ because it enforces the variable `f` to be of type `label_fun`. The existing data item `((Actor a', as), n)` is replaced by `f ⇳((Actor a', as), n)` while preserving the label `(Actor a', as)` owing to the properties of type `label_fun`. Clearly, the actor needs to be `eval` enabled in his location where also the data must reside.

```
process:
G = graphI I ⟹ a @_G l ⟹
enables I l (Actor a) eval ⟹
((Actor a', as), n) ∈ snd (lgra G l) ⟹
Actor a ∈ as ⟹
I' = Infrastructure
     (Lgraph (gra G)(agra G)(cgra G)
       ((lgra G)(l := (fst (lgra G l),
       snd (lgra G l)  - {((Actor a', as), n)}
             ∪ {f ⇳ ((Actor a', as), n)}))))
      (delta I)
⟹ I →_n I'
```

*4) The delete rule:* enables the owner of the data to delete his or her data from a location in the infrastructure graph. Note that, different to the previous rules, here are no preconditions on the location of the actor nor the location of the data other than that they are in the infrastructure graph. Neither is there any requested enabledness of actions imposed on the actor. That is, the owner can delete his data anywhere.

```
del_data:
G = graphI I ⟹ a ∈ actors G ⟹
l ∈ nodes G ⟹
((Actor a, as), n) ∈ snd (lgra G l) ⟹
I' = Infrastructure
     (Lgraph (gra G)(agra G)(cgra G)
       ((lgra G)(l := (fst (lgra G l),
           snd (lgra G l) - {((Actor a', as), n)}))))
      (delta I)
⟹ I →_n I'
```

## V. Application Example from IoT Healthcare

### A. System Model

The example of an IoT healthcare systems is from the CHIST-ERA project SUCCESS [4] on monitoring Alzheimer's patients. Figure 2 illustrates the system architecture where data collected by sensors in the home or via a smart phone helps monitoring bio markers of the patient. The data collection is in a cloud based server to enable hospitals (or scientific institutions) to access the data which is controlled via the smart phone. We show the encoding of the `igraph` for this system

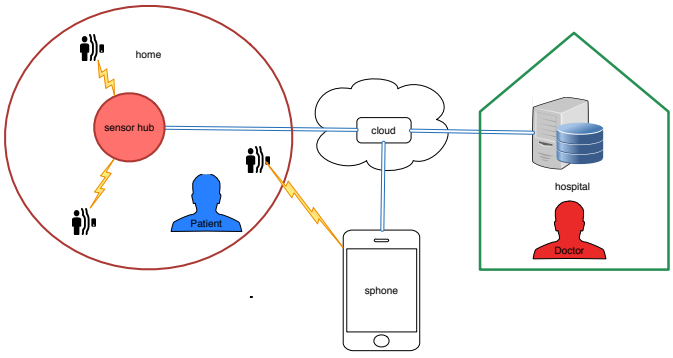


Fig. 2. IoT healthcare monitoring system for SUCCESS project

architecture in the Infrastructure model.

```
ex_graph ≡ Lgraph
   {(home, cloud), (sphone, cloud), (cloud,hospital)}
   (λ x. if x = home then {''Patient''} else
         (if x = hospital then {''Doctor''} else {}))
   ex_creds ex_locs
```

The data and its privacy access control definition is given by the parameter ex_locs specifying that the data 42, for example some bio marker's value, is owned by the patient and can be read by the doctor.

```
ex_locs ≡ (λ x.  if x = cloud then (''free'',
{((Actor''Patient'',{Actor''Doctor''}),42)}) else {})
```

### B. Policy Enforcement

Using the formal model of infrastructures, we can now prove privacy by design for GDPR compliance of the specified system. We show how the properties relating to data ownership, processing and deletion can be formally captured using Kripke structures and CTL and the Infrastructure framework.

*1) Owner and listed readers can access:* we can now use the basic definition over the DLM labels from Section IV-A to express that in all states globally (AG) owners and listed readers can access a data item.

```
theorem GDPR_one:
 ∀ h ∈ gdpr_actors. ∀ l ∈ gdpr_locations.
   Actor h ∈ {owner d} ∪ readers d ⟶
   gdpr_Kripke ⊢
        AG {x. has_access (graphI x) l (Actor h) d }
```

*2) Owner can delete:* the property that any actor of the infrastructure can delete his or her data at any time is expressed using the definition actor_can_delete which expresses that a location l does not contain any data for an actor h.

```
actor_can_delete I h l ≡
 (∀ as n. ((h, as), n) ∉ (snd (lgra (graphI I) l)))
```

This definition is then used within an AG statement that entails another temporal formula: for all paths holds in all states that there is a path x in which the data of Actor h is deleted.

```
theorem GDPR_two:
 ∀ h ∈ gdpr_actors. ∀ l ∈ gdpr_locations.
   gdpr_Kripke ⊢
      AG ( EX {x. actor_can_delete x (Actor h) l})
```

The inner formula is an EX formula (instead of an AX) since not necessarily in all possible next states from the state x the data is deleted: Actor h may still consent to keep the data. We have formulated this property here within the healthcare application but it can be proved in more general terms – for arbitrary Kripke structures – at the general level of the theory Infrastructure.

*3) Processing preserves privacy:* we can prove that processing preserves ownership as defined in the initial state for all paths globally (AG) within the Kripke structure and in all locations of the graph. Note, that it would not be possible to express such a set using a universally quantified formula within a temporal operator when using Modelcheckers since they only allow propositional logic within states. This generalisation is only possible since we use Higher Order Logic.

```
theorem GDPR_three: h ∈ gdpr_actors ⟹
  l ∈ gdpr_locations ⟹
 owns (Igraph gdpr_scenario) l (Actor h) d ⟹
 gdpr_Kripke ⊢
    AG {x. ∀ l ∈ gdpr_locations.
         owns (Igraph x) l (Actor h) d }
```

## VI. Conclusions

In this paper, we have summarised the GDPR and demonstrated that the Isabelle Infrastructure framework can be used to apply this data protection regulation for compliance checking. We have illustrated how the Isabelle Infrastructure framework can formally encode the technical requirements entailed in the GDPR by using Kripke structures and the temporal logic CTL. We have illustrated on an IoT healthcare patient monitoring system, how to encode the system architecture and how to specify the privacy access control. Finally, we have formalized the major proof obligations given by the GDPR.

The use of CTL is reminiscent of Modelchecking techniques. However, the use of Modelchecking is restricted owing to the state explosion problem to finite domains and propositional logic in the states. Using Isabelle, we overcome this restriction and enable use of quantification and induction necessary for invariant proofs (like the preservation property (3) (Section V-B3)). For legal applications in computer science deontic logics have been experimented with (see [14]) for an overview). However, to our knowledge, this is the first application of Isabelle or HOL to modeling GDPR. A major advantage of using a proof assistant is that the formal specification and proofs are machine-checked. Furthermore, abstract system specifications can be provably refined and finally code can be extracted to major programming languages, e.g. Scala.

## References


[1] E. Union, "The eu general data protection regulation (gdpr)," Accessed 20.3. 2018, http://www.eugdpr.org.

[2] F. Kammüller and C. W. Probst, "Modeling and verification of insider threats using logical analysis," *IEEE Systems Journal* vol. 11, no. 2, pp. 534–545, 2017.

[3] A. C. Myers and B. Liskov, "Complete, safe information flow with decentralized labels," in *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 1999.

[4] CHIST-ERA, "Success: Secure accessibility for the internet of things," 2016, http://www.chistera.eu/projects/success.

[5] D. M. Cappelli, A. P. Moore, and R. F. Trzeciak, *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)*, 1st ed., ser. SEI Series in Software Engineering. Addison-Wesley Professional, Feb. 2012.

[6] F. Kammüller and M. Kerber, "Investigating airplane safety and security against insider threats using logical modeling," in *IEEE Security and Privacy Workshops, WRIT'16*. IEEE, 2016.

[7] F. Kammüller, M. Kerber, and C. Probst, "Towards formal analysis of insider threats for auctions," in *8th ACM CCS International Workshop on Managing Insider Security Threats, MIST16*. ACM, 2016.

[8] F. Kammüller, J. R. C. Nurse, and C. W. Probst, "Attack tree analysis for insider threats on the IoT using Isabelle," in *Human Aspects of Information Security, Privacy, and Trust*, LNCS Springer, 2016.

[9] F. Kammüller, "Human centric security and privacy for the iot using formal techniques," in *3d International Conference on Human Factors in Cybersecurity*, vol. 593, AISC, Springer 2017.

[10] ——, "Formal models of human factors for security and privacy," in *5th International Conference on Human Aspects of Security, Privacy and Trust, HCII-HAS 2017*, LNCS **10292**, Springer, 2017.

[11] ——, "Isabelle modelchecking for insider threats," in *Data Privacy Management, DPM16, assoc. w. ESORICS'16* LNCS **9963**, 2016.

[12] ——, "A proof calculus for attack trees," in *Data Privacy Management, DPM17, assoc. w. ESORICS'17*, LNCS **10436**, Springer, 2017.

[13] ——, "Isabelle infrastructure framework with iot healthcare s&p application," 2018, available at https://github.com/flokam/IsabelleAT.

[14] J. C. Meyer, R. J. Wieringa, and I. W. on Deontic Logic in Computer Science, *Deontic Logic in Computer Science Normative System Specification*, 1993.