Elicitation of Privacy Requirements for the Internet of Things Using ACCESSORS

Christoph Stach and Bernhard Mitschang

Institute for Parallel and Distributed Systems, University of Stuttgart, Universitätsstraße 38, D-70569 Stuttgart, Germany
`{stachch,mitsch}@ipvs.uni-stuttgart.de`

Abstract. Novel *smart* devices are equipped with various sensors to capture context data. The *Internet of Things* (*IoT*) connects these devices with each other in order to bring together data from various domains. Due to the IoT, new application areas come up continuously. For instance, the quality of life and living can be significantly improved by installing connected and remote-controlled devices in *Smart Homes*. Or the treatment of chronic diseases can be made more convenient for both, patients and physicians, by using *Smart Health* technologies.
For this, however, a large amount of data has to be collected, shared, and combined. This gathered data provides detailed insights into the user of the devices. Therefore, privacy is a key issue for such IoT applications. As current privacy systems for mobile devices focus on a single device only, they cannot be applied to a distributed and highly interconnected environment as the IoT. Therefore, we determine the special requirements towards a permission models for the IoT. Based on this requirements specification, we introduce ACCESSORS, a data-centric permission model for the IoT and describe how to apply such a model to two promising privacy systems for the IoT, namely the *Privacy Management Platform* (*PMP*) and *PATRON*.

Keywords: Permission Model · Data-Centric · Derivation Transparent · Fine-Grained · Context-Sensitive · Internet of Things · PMP · PATRON.

1 Introduction

Today there is a trend to equip everyday objects, such as wristwatches, with a variety of sensors. Due to new, low-cost and power-saving connection standards, these devices can also be easily interconnected. Due to their versatility and easy handling, these *Things*[1] get into the focus of the general public [4]. As a result, the so-called *Internet of Things* (*IoT*) is becoming increasingly popular [14]. This opens up a wide range of possible application scenarios for the IoT, including *Smart Homes* [9], *Smart Health* [50], and *Smart Cars* [62].

These versatile fields of application are facilitated by the two key characteristics of the Things: On the one hand, the built-in sensors are able to capture any

[1] We use the term "Thing" for any device equipped with sensors and Internet access.

kind of context information, such as location data, surrounding sounds, or even health data. As these Things are common everyday objects, they are no longer perceived by the user as computers and are always carried along naturally [67]. This makes it possible to capture data about users on a permanent basis.

On the other hand, the Things are interconnected. As a result, they are able to exchange autonomously the captured data with each other [31]. Thus, it is sufficient if only a limited number of sensors are installed in each Thing in order to gain comprehensive contextual knowledge about their users. This knowledge can be used in IoT applications (or *apps*) to adapt their functionalities to their users' private lives. By this, IoT apps are able to predict the most likely user demands in the prevailing situation and provide the currently most beneficial services [36]. So, they contribute to improving the quality of life.

The IoT apps are also not constrained by the usually limited computing power of the Things. By transmitting the data to the *Cloud* (or upstream components, such as *Fog Instances*), IoT apps have access to virtually unlimited resources in terms of computing power, memory, or storage. Studies show that, despite this transmission, data processing can be realized in almost real-time [44].

However, these unlimited processing possibilities result in new threat scenarios [35]. Machine learning techniques can be applied to IoT apps in order to detect connections between existing data sources and derive more knowledge from the available data [29]. Users are extremely worried about the overwhelming potential of these apps [11]. Individuals cannot only be monitored permanently without their knowledge, but also additional information about them can be generated from the collected data. Therefore, privacy has to be a key issue for any IoT app [1].

While there are unambiguous regulations for the processing of personal data from a legal point of view (e. g., the European General Data Protection Regulation [64]), there is a lack of technical approaches for the implementation of comprehensive privacy mechanisms for the IoT [26]. In this respect, it is important that the entire IoT app is taken into account, i. e., effective privacy mechanisms have to be applied to both, the Things as well as their back-ends [53]. Yet, even simple privacy management systems, i. e., systems that restrict access to a certain data processing unit, overwhelm users already [20]. Moreover, users don't know which information can be derived from which data [41] and whether this information poses a privacy threat [19]. For instance, a proximity sensor can disclose the absolute location of a user also, when it gathers the distance to a Thing with a stationary location [24].

For this very reason, we introduce a data-centric and thus comprehensible privacy approach for the IoT, tackling both, Things as well as their back-ends. To that end, we provide the following five contributions in our work:

(1) We deduce requirements towards a permission model for IoT apps from a use case scenario.
(2) We analyze permission models which are applied in existing privacy systems and provide a comprehensive overview of their features and their applicability in the IoT domain.

(3) We construct a d**a**ta-**c**entri**c** p**e**rmi**ss**ion m**o**del for the Inte**r**net of Thing**s**, called ACCESSORS.
(4) We apply ACCESSORS to both, mobile devices (*PMP* [56, 57]) and distributed stream processing systems (*PATRON* [54, 55]), that is, the back-end of IoT apps. However, we could use any of the many similar privacy systems as a foundation for our model without a loss of argument.
(5) We evaluate our model and assess its utility.

This paper is the extended and revised version of the paper entitled "ACCESSORS: A Data-Centric Permission Model for the Internet of Things" [58] presented at the 4th International Conference on Information Systems Security and Privacy (ICISSP) 2018. This extended paper considers all layers of IoT apps, whereas the original paper focuses at the Sensor and Smartphone Layer, only.

The remainder of this paper is as follows: Section 2 introduces a real-world use case scenario to illustrate the challenges for a permission model for IoT apps. Then, Sect. 3 postulates five key requirements for such a permission model. Section 4 discusses various existing permission models. Our model—ACCESSORS—is introduced in Sect. 5. Section 6 describes how to apply ACCESSORS to a privacy system. Finally, Sect. 7 assesses our approach before Sect. 8 concludes this work and gives a short outlook on future work.

2 Use Case Scenario

The application of IoT technologies for non-invasive glucose level sensing and diabetes management is described by Istepanian et al. [28]. Figure 1 shows the architecture of such an application. Various sensors (e. g., a glucose meter) initially record a wide range of health data (e. g., blood sugar level) at the *Sensor Layer*. The measurement data of an individual patient is transmitted to his or her Smartphone and consolidated at the *Smartphone Layer* [61]. However, patients do not know exactly what data is being passed on, especially since such a device is capable of collecting different types of data—e. g., some devices add location data to any glucose measurement, as this information might be relevant for later diagnostic analyses [33, 52, 59].

The *Back-End Layer* accumulates the data of several patients (e. g., grouped by the attending physician) at a central server in order to enable comprehensive analyses [8]. By combining the gathered data, further knowledge can be derived. For instance, a combination of blood sugar values and location data enables to draw inferences about the user's eating behavior as a rising blood sugar level shortly after walking past a candy shop indicates that the user has bought some sweets [32]. The *Presentation Layer* provides tools to present the results to health professionals. However, patients have neither insight into which data is collected at the back-end nor which information can be derived from it via data mining.

Despite this insecurity concerning the processing of sensitive data, the number of IoT health devices available is growing significantly. With each generation more sensors and a larger range of features are introduced [65]. The accuracy of

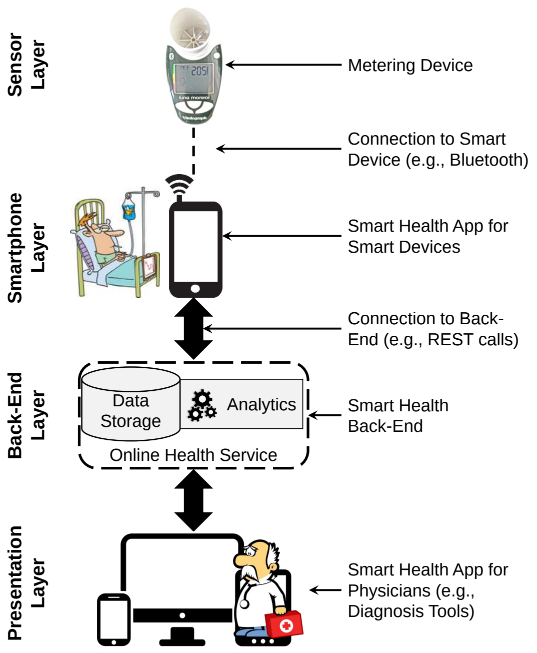
Sensor Layer
Smartphone Layer
Back-End Layer
Presentation Layer
Metering Device
Connection to Smart Device (e.g., Bluetooth)
Smart Health App for Smart Devices
Connection to Back-End (e.g., REST calls)
Data Storage
Analytics
Online Health Service
Smart Health Back-End
Smart Health App for Physicians (e.g., Diagnosis Tools)

Fig. 1. Architecture of an IoT Health Application [60].

the sensors is also improving [34]. But not every IoT app requires this high level of accuracy. This implies that from a privacy perspective, data quality should be downgraded in order to conceal private information which is not required for the app to run. While some of the data provided by such IoT devices is uncritical from a privacy point of view or so vital that the data is required all the time, other sensitive data is required only in case of an emergency. For instance, in case of an insulin shock, health has priority over privacy and thus, any available data should be sent to the physician to provide the best possible medical attendance.

Such a scenario requires a privacy system—or more precisely its permission model—to meet several novel requirements in order to be effective [49]. For instance, focusing on data-centric protection goals is becoming increasingly important [2]. This is further amplified by the fact that IoT apps assemble its data from various sources, some of them even unknown to the user [63]. Moreover, since new devices with all-new sensors are constantly being released, the permission model must adapt to such an evolving environment [1].

3 Requirements Specification

As the scenario given in Sect. 2 shows, the IoT defines some novel requirements towards a permission model, that are detailed in the following.

[R1] Data-Centric Policy Rules. To be understandable and manageable for the user, the permissions have to refer to types of data (e. g., blood sugar level) instead of data providers (e. g., glucometer). Although it is evident that a glucometer measures the blood sugar level, some devices are also able to capture location data. If policy rules are solely based on data providers, a user might allow a health app to use his or her glucometer without knowing that s/he also gave access to non-medical data (e. g., location data) in the process. The same type of data can even be provided by several devices (e. g., the blood sugar level is provided by glucometers and *Apple Watches*). If a user wants to prohibit access to this data, then a respective rule has to be applied to any possible provider.

[R2] Derivation Transparency. An IoT app has access to various types of data via several sensors. However, by combining this data, new information can be derived. Such coherences have to be representable by a permission model. For instance, if A can be derived from B and C and a user prohibits access to A, then an app must not be allowed to access B and C at the same time. This can be archived by describing what information can be derived from which sources. The user can then assign permissions at data level and the privacy system must apply appropriate rules to the respective sources.

[R3] Extendable Permission Model. The IoT is constantly evolving as new sensor technologies or communication standards emerge. A static permission model, that is, a model with a fixed set of protected entities, quickly becomes obsolete. Therefore, the model must be dynamically extendable. In particular, all extensions must be backward compatible, i. e., the extension of the model must not invalidate previous rules.

[R4] Fine-Grained Policy Rules. In order to give a user the opportunity to manage his or her data confidentially, s/he needs full control over the distribution and dissemination of information. This means that the permission model has to support fine-grained policy rules in two respects: On the one hand, the protected entities have to be fine-grained. For instance, Android provides a `Bluetooth` permission which restricts access to any device connected via Bluetooth. Yet, this permission does not address a specific type of data or sensor. As a consequence, users have to permit apps to use a Bluetooth headphone and a Bluetooth medical device at the same time via a single permission. On the other hand, a user has to have several choices how to constrain a certain permission. Most permission models follow a binary logic, only (grant or deny). However, a permission for location data also could restrict the accuracy of the data.

[R5] Context-Sensitive Policy Rules. Since IoT apps are often context-aware, i. e., an app reacts on the situation it is currently used, the policy rules should be context-sensitive as well. For instance, a medical app should have access to any kind of data in case of an emergency. Otherwise, more restrictive policy rules should be applied. Dey defines context as "any information that can be used to characterize the situation of an entity" [17].

4 Related Work

Based on these requirements, we analyze permission models which are currently used in the IoT context. In the following, we differentiate between privacy solutions for the Back-End Layer (Sect. 4.1) and privacy solutions for the Smartphone Layer (Sect. 4.2). The other two layers of an IoT app do not have to be considered specifically, since no data processing is done here, i. e., privacy solutions for the Smartphone and Back-End Layer also cover the privacy issues of these two layers.

4.1 Privacy Solutions for the Back-End Layer

Several methods have been developed to control the information flow in stream processing [68] and event processing systems [25]. These systems can also be applied to analyses performed at the Back-End Layer of an IoT app. Most access control mechanisms currently existing for these systems such as *DEFCON* [37] are attribute-based. That is, these systems ensure that certain attributes (in the stream of events) are only visible to authorized processing operators. However, this is overly restrictive since it implies that operators either always have access to certain attributes or never. In other words, the underlying permission model simply assigns to each attribute either the label "granted" or "denied". Some stream processing systems such as *ACStream* [10] provide context-based access control. That is, they add to each attribute permission pair also information about the context under which the respective privacy rule should be applied. In other words, these triples allow a more fine-grained access control to information. Yet, the access is still controlled at the level of attributes. A different approach towards privacy for the Back-End Layer called PATRON is introduced in Sect. 6.2.

However, all these approaches are not designed for the end user, but for IT specialists. Therefore it is also not possible to enforce individual privacy rules for each user with these approaches, as it is the focus of this paper. Rather, it is intended to provide a simple way to regulate access to vast amounts of data in accordance with a general policy. Therefore, all of the applied simple permission models are not suitable for our purposes.

4.2 Privacy Solutions for the Smartphone Layer

Due to the highly heterogeneous IoT landscape and the various operating systems available for the Things, a lot of different privacy systems and thus permission models are being used at the Smartphone Layer. Yet, there are several efforts to establish Android as the key operating system for the IoT, e. g., *Android Things* [22] or *RTAndroid*[2] [30]. Although we focus on Android-based privacy systems in our work, the findings can be applied to any mobile platform, as they also use comparable permission-based privacy systems [7].

Android applies a quite simple permission model (see Fig. 2). Each Permission regulates either the usage of a system functions (e. g., adding entries to the

[2] A refined version of RTAndroid called *emteria.OS* is available at https://emteria.com.

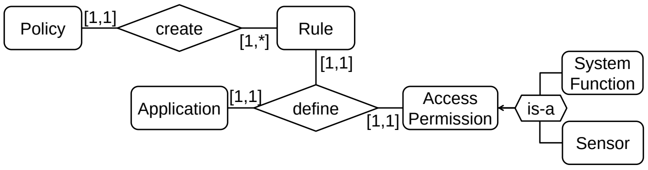

Fig. 2. The Permission Model Applied in Android [58].

calendar) or access to a sensor (e. g., the camera). An app has to request the appropriate permissions before it is able to use such a resource. For *normal* permissions a policy rule is created when an app is installed (i. e., they are automatically granted), while *dangerous* permissions have to be granted at runtime [18].

When new permission types are added or existing permissions are relabeled, app developers have to add these permissions to their already released apps in order to keep them operative [48]. Yet, several system functions and sensors can be controlled by a single permission. This makes it very hard for users to comprehend the permissions [20]. Moreover, there are so many different permissions right now (even for noncritical operations such as the usage of the vibration function) [20] which makes it even harder to grasp the permissions. As a consequence, Google no longer informs about noncritical permissions. However, Google classifies even access to the Internet or the usage of Bluetooth device as noncritical operations (see Fig. 3). Yet, both can have a severe impact on the user's privacy. Thus, such a basic permission model is not applicable for the IoT.

Sekar et al. [47] introduce *Selective Permissions*. This means that every Android permission requested by an app is stored in a *Shadow Manifest* that can be changed at runtime. This allows a user to revoke certain permissions similar to Android runtime permissions. However, Selective Permissions have two advantages. On the one hand, a user can revoke any permission; on the other hand, a missing permission does not lead to a security exception. Instead, a null value is returned to the app. However, this approach does not change the Android permission model and therefore does not meet any of the requirements defined in Sect. 3.

CRêPE introduces a context-sensitive permission model [12]. Each access to a data source, i. e., each permission request, can be linked to a spatio-temporal context. This context defines a condition under which the permission is granted. However, the rules are mapped to Android permissions and therefore CRêPE has the same shortcomings as the Android permission model.

Apex introduces an XML-based policy language to restrict the use of Android permissions [39]. For instance, the user can define how often a particular permission can be used or in which chronological order permissions can be assigned. Apart from such constraints, the permission model does not allow extensive contextual constraints or fine-grained permission settings. Furthermore, the model is

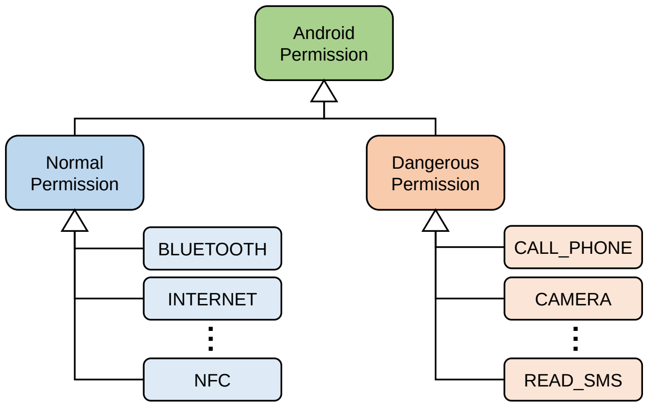

Fig. 3. Classification of Android Permissions [based on 58, 66].

neither data-centric nor derivable and it cannot be extended because it is based on Android permissions.

In *YAASE*, a user defines which operations a particular application may perform on a resource, that is, either a content provider or a service provider [43]. Data from these resources can be tagged,, for example, to distinguish between public and private data provided by the same resource. The user defines whether only resources with a certain tag are accessible for an app. S/he can also define operations that must be performed before the data is forwarded to an app, such as a filer operation to remove sensitive data. In this way YAASE is able to define very fine-grained policy rules. However, these rules are not data-centric, transparent or context-sensitive. In addition, the extensibility of the model is limited to specified operations.

Sorbet addresses the unrestricted information flows between apps [21]. Therefore, the underlying permission model allows to specify information-flow constraints to prevent privilege escalation, i. e., the transfer of permissions between apps. This can also be used to introduce a kind of context-sensitivity in Sorbet. Furthermore, the Sorbet is able to protect any kind of component (e. g., services or content providers). So, its permission model is extendable. Moreover, it is possible to define constraints in the model to limit the usage of certain permissions, e. g., by adding a lifespan to it. This also reduces the risk of privilege escalation. Therefore, Sorbet has fine-grained, yet Boolean policy rules. Also, Sorbet neither supports data-centric nor derivation transparent policy rules.

RetroSkeleton introduces an app rewriting system. So, it is able to replace method calls with arbitrary code fragments [15, 16]. The replacement rules are specified as Clojure command sequences. The user can draw on the full expression power of Closure to define derivation transparent, fine-grained, and context-sensitive policy rules—provided that s/he has the required skills. As the model is generic and does not rely on preexisting permissions, it is extendable. Yet, as it only replaces method calls, RetroSkeleton's permission model is not data-centric.

Constroid grants subjects (e. g., processes) the rights to process data items (e. g., all business contacts) [45]. Constroid relies on the $UCON_{ABC}$ model [40]. Each data item can be associated with attributes (e. g., contacts without a private phone number) to restrict the access rights. As access rights are linked to data, Constroid considers only create, read, update, and delete operations. Optional conditions specify whether a rule is applicable under a certain context. Yet, the model is not extendable and does not support derivation transparent policy rules.

SPoX is a specification language for security policies [23]. SPoX rules define a state machine that accepts all command sequences that comply with the security policy. Backes et al. [5] use this language in their data protection system *AppGuard*. This enables the user to formulate fine-grained policy rules, e. g., by limiting network access to a specific address. AppGuard's permission model is extensible, because each command can be restricted by policy rules. Thus, new data sources are also supported by AppGuard out of the box. By linking several rules, the user is able to model a kind of derivation transparency. The context in which a particular command is executed can also be restricted [6]. However, these restrictions only apply to the command sequence and not to the context of the user. Furthermore, the data protection model is not data-centric.

Scoccia et al. [46] introduce flexible permissions for Android called *AFP*. In AFP, permissions are assigned to features of an app. This means that an app may only request a permission to perform a specific task. Furthermore, AFP enables the assignment of fine-grained permissions, e. g., by granting access only to selected contacts instead of the entire contact list. Since AFP defines its own permissions, the model is extensible. Nevertheless, the policy rules are neither data-centric nor derivation transparent.

DroidForce introduces data-centric policy rules [42]. *OSL* [27] is used to specify the rules. This enables users to add temporal conditions as well as cardinality constraints and time constraints to each permission. Therefore, both fine-grained and context-sensitive rules are supported. The main feature of DroidForce is its focus on data-centric permissions. This means that the permissions are mapped to data domains (e. g., location data or contact data) and not to sensors or system functions. However, relationships between protected data sources cannot be modeled and the model used cannot be extended.

The *Privacy Management Platform* (*PMP*) [57] introduces the *Privacy Policy Model* (*PPM*) [56] (see Fig. 4). The PPM is extendable and enables fine-grained and context-sensitive policy rules. Therefore, it defines so-called *Service Features*. These are self-contained fragments of an app which can be (de-)activated in order to meet the users' demands. That way, permissions can be directly granted to specific Service Features. Each permission is restricted to a certain purpose to reflect the privacy requirements as good as possible. *Resources* manage the access to data sources or sinks. Related Resources can be pooled in a *Resource Group* (e. g., GPS and WiFi positioning are part of a location Resource Group). So-called *Privacy Settings* can be defined for each Resource (e. g., to reduce the accuracy of location data for a certain Service Feature). The Resources can also be used to define contextual constraints. These constraints specify a scope of

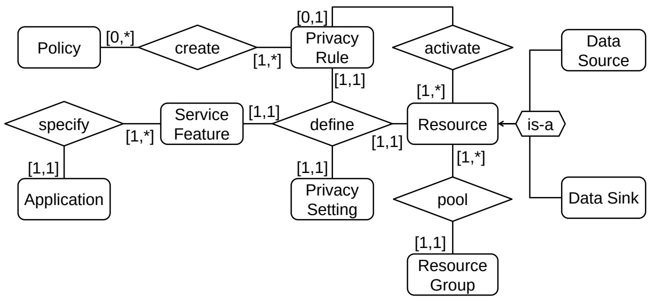

Fig. 4. The Privacy Policy Model (PPM) [based on 56, 58].

application for each privacy rule. Due to these features, the PPM meets most of the requirements towards a permission model for the IoT. However, the missing support of data-centric policy rules overstrains users unjustifiably. The following *Smart Health* example illustrates this issue:

If a user manages his or her electronic health data record on his or her Smartphone, s/he can use a *Smart Health* app. However, s/he only wants this app to gather certain health data, e. g., his or her fitness progress including heart rate (pulse meter), activities (accelerometer and orientation sensor), and training locations (GPS). Additionally, s/he wants to use the camera of the Smartphone for a visual documentation of his or her training progress. S/he could use this electronic health data record to get a special tariff rate from his or her insurance company in which a healthy lifestyle is rewarded.

It is obvious that such a user does not want to share any additional data with his or her insurance company which might indicate an illness (e. g., a high body temperature), as this could lead to a higher insurance rate. The PPM enables to prohibit the *Smart Health* app to access a Bluetooth medical thermometer for the purpose of measuring the body temperature. The thermometer then is represented as a Resource and the measuring is represented as a Service Feature.

However, if we assume that the Smartphone is equipped with a thermographic camera, the user must be familiar with this feature. If s/he does not consider this functionality of his or her device, such a camera can also display the body temperature. To prevent this, s/he must define an additional privacy rule in the PPM for the measuring Service Feature that prohibits access to the camera Resource. For each data source, s/he must reflect what knowledge can be derived from his data.

Although the PPM is able to secure sensitive data in IoT apps, with an increasing number of different sensors, it is almost impossible for a user to keep track of all the possible data leaks due to the Resource-centric Policy Rules of the PPM. Nevertheless, the PPM is a sound foundation for ACCESSORS.

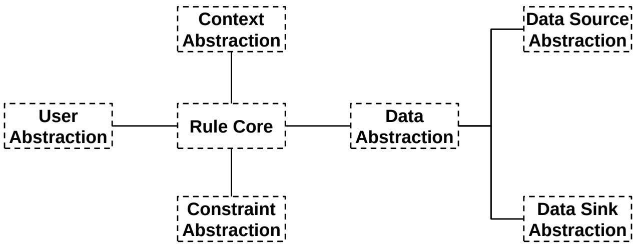

Fig. 5. Basic Structure of an ACCESSORS Permission.

5 The ACCESSORS Model

The study of the related work shows that there is currently no suitable permission model for IoT apps. Exisiting models are too superficial and general-purposed. However, the PPM is well suited for the Smartphone Layer of IoT applications as long as a manageable number of sensors are involved. In an IoT scenario with many different sensors, a different approach is required because humans are used to think data-centric. This means that a user knows what data s/he wants to conceal and s/he does not want to worry about which sensor or data source could disclose this type of data. Current approaches, by contrast, require a separate rule for each data source that contains this information. For this purpose, a permission model must be able to map data producers to the type of data they provide. In this way, the user can select the type of data to be made available to an app (e. g., body temperature) and the model unfolds, which data sources must be considered (e. g., medical thermometer and thermographic camera). Our approach of a d**a**ta-**c**entri**c** p**e**rmi**ss**ion m**o**del for the Inte**r**net of Thing**s**—ACCESSORS for short—achieves this by introducing six abstraction levels.

Figure 5 shows the basic structure of an ACCESSORS permission. In the following, we detail its seven key components (*Rule Core*, *User Abstraction*, *Data Abstraction*, *Data Sink Abstraction*, *Data Source Abstraction*, *Context Abstraction*, and *Constraint Abstraction*) and elaborate on how they contribute to meet the five requirements, specified in Sect. 3.

Rule Core. Similar to the PPM, an ACCESSORS policy rule essentially has three main parts: an *access purpose*, a permission to *access* a data processing unit[3], and a *constraint*. These triplets form the *rule core*. Optionally, each policy rule can be associated with a *context* in which it is activated (see Paragraph Context Abstraction).

User Abstraction. Each *inquiring entity*—i. e., an *app*, a *Smart Thing*, or even a *user*—can specify one or more access purposes that require access to a protected

[3] A data processing unit is either a *data producer* or a *data consumer* (see Paragraph Data Abstraction).

type of data. An access purpose is a code fragment within an app that performs a single task. For example, such an access purpose in the use case scenario described in Sect. 2 could be the graphical representation of all locations on a map where the user measured his or her blood sugar level. In this way, the permissions are not granted to an app in general, but they are valid for a specific access purpose, only. As a consequence, a user can decide which access is justified for a particular type of app and if s/he is willing to grant the specified access rights for the offered service. Similar to PPM, non-essential app features can be skipped to reduce the amount of required private data. *User abstraction* ensures that other types of Smart Things can be added as needed.

Data Abstraction. Data abstraction enables the linking of permissions with both, *data producers* and *data consumers*. However, the focus for both units is on the type of data that is produced or consumed. This means that an inquiring entity must indicate which data it requests access to, e. g., location data or health data, instead of a specific data processing unit such as GPS or a glucometer.

Data Sink Abstraction. Every data consumer is linked to multiple *data sinks* such as *apps* or *services*, *data stores* or other *Smart Things*. That is, the user can set policy rules on how data can be preprocessed for an app. For example, s/he could allow an app to use a service that stores health data for long-term monitoring of a particular health condition.

Data Source Abstraction. Each data producer is associated with a certain type of *information*. Information is any aspect that can be derived from raw data. This means that it can be the raw data itself (e. g., a single blood glucose level metering) or any other type of higher order data obtained by combining several sources (e. g., a health record with data from different meters). Different *data sources* can be specified for each data type in the ACCESSORS model. A data source does not necessarily have to be a *sensor*, but *apps*, *data storages*, and *Smart Things* are also qualified as data sources. In this way, complex relationships can be modeled (e. g., the information "activity" can be derived either by a combination of data from an accelerometer and a position sensor or directly by readings from a fitness tracker). By *data sink abstraction* and *data sources abstraction*, the policy rules remain completely detached from a specific technology. The rules are automatically adapted to the available data sources and sinks accordingly.

Context Abstraction. An activation context can optionally be assigned to each policy rule. This context describes the conditions under which a rule must be enforced by a privacy system. In accordance with Dey [17], we describe the context as a *spatio-temporal* condition (e. g., a certain rule should only be applied during working hours) or as a higher order *situation* (e. g., a certain rule only applies in the case of a medical emergency). Higher order situations can be modeled as a sequence of values provided by data producers.

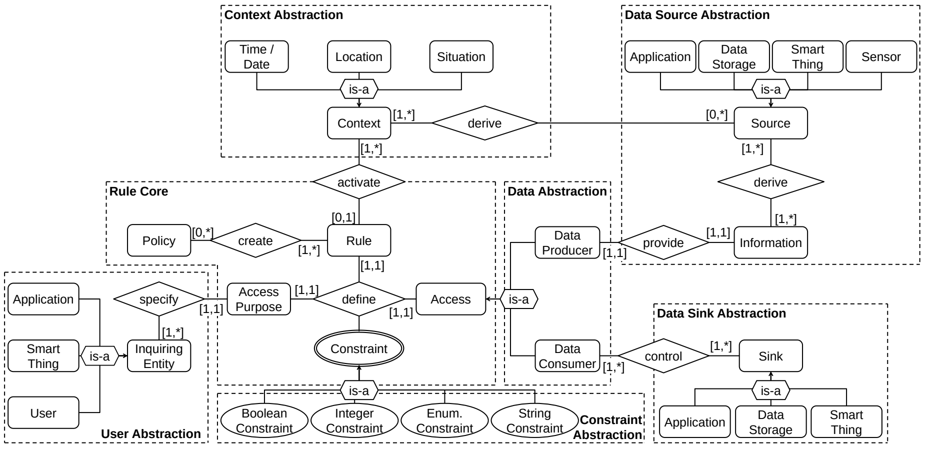

Fig. 6. The Data-Centric Permission Model for the Internet of Things [58].

Constraint Abstraction. Different constraints can be defined for each rule. The most fundamental constraint is a Boolean constraint to grant or deny access to certain type of data[4]. Depending on the type of data, ACCESSORS supports three additional constraint types. Integer conditions can be used to define an upper or lower limit. For example, maximum accuracy for a particular type of data, such as location data, can be specified in this way. An enumeration constraint defines several valid setting options. For example, for medical records, there may be settings that only allow access to domain-specific data records such as pulmonary data or cardiac data. Finally, string constraints allows to enter textual conditions. For example, a user can specify a MAC address of a Thing with which s/he wants to share his or her data. This ensures that the health data is only sent to the specified destination address.

Figure 6 shows the detailed ACCESSORS model with all components of the Rule Core and the six abstraction layers. Overall, ACCESSORS supports data-centric policy rules, since the focus of the permissions is on data types instead of actual data processing units. Since data producers provide higher order information, which can be composed of data from several sources, ACCESSORS is able to model relationships between different types of data and sources. Since the policy rules only link access purposes, access permissions, constraints, and contexts, they are independent of specific inquiring entities or data producers / data consumers. This means that ACCESSORS has two types of extensibility due to its abstraction layers. On the one hand it can be extended (e. g., by adding new Things) and on the other hand it can be advanced (e. g., by adding new relationships between data sources when new methods to derive a certain kind of information from raw data are discovered). Policy rules modeled with ACCESSORS are highly fine-grained. On the one hand, the multi-value constraints

[4] If the access permission is denied, the particular code fragment is skipped in the app.

enable highly precise fine-tuning of permission rights. On the other hand, since the permissions are bound to a certain access purpose and do not have to be granted to an app in total, the user can tailor the privacy policy precisely to his or her needs. Each policy rule can be enriched by an activation context. This context is generic, as it can be composed of all currently available data sources.

A comparison of the PPM (see Fig. 4) with ACCESSORS (see Fig. 6) shows that the two models have several common components. The rule core of ACCESSORS almost matches the PPM. However, ACCESSORS introduces additional abstraction layers for users, data, data sinks, data sources, contexts, and constraints. Furthermore, ACCESSORS takes a different protection goal into account. While the PPM is designed for Smartphones and therefore only considers apps as potential attackers and sensors or system functions as possible targets (labeled as Resources), ACCESSORS is outright designed for the IoT. For this reason, not only the potential attackers are interpreted in the broader sense (inquiring entities such as apps, Smart Things, or users), but also concerning the protected targets, ACCESSORS has a different focus. The targets are tailored to the types of data instead of data sinks or data sources.

Nevertheless, it appears to be obvious to map the policy rules defined in ACCESSORS to PPM rules due to their great similarities. Moreover, as the PPM is already applied to an existing privacy systems for the Smartphone Layer, the PMP, we can use this infrastructure to enforce the ACCESSORS rules as well. The following section describes how to map ACCESSORS policy rules to PPM rules. Furthermore, we show how ACCESSORS can also be used in a privacy system for the Back-End Layer. To this end, we introduce PATRON, a privacy mechanism for stream processing systems. PATRON focuses at two goals: On the one hand, it hides private information from unauthorized parties and on the other hand, it ensures quality of service of the controlled IoT apps. ACCESSORS is a great support in achieving these goals.

6 Application of ACCESSORS in IoT Privacy Systems

As ACCESSORS is not committed to a certain privacy system, it can be applied to any given privacy system in order to control access to any kind of private data. Due to its similarities to the PPM, the usage of the model in a privacy system for end devices is the most reasonable use case. This case of application is described in Sect. 6.1 using the example of the PMP.

However, ACCESSORS can also be used to identify the privacy demands of users. Due to the systematic yet human comprehensible notation of ACCESSORS, it is also possible to express complex correlations between gathered data and derivable knowledge in an automated processable way. This enables end-users to configure privacy systems for the Back-End Layer of IoT apps. This is described using the example of PATRON in Sect. 6.2.

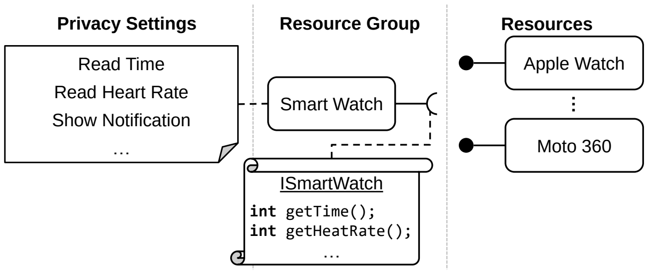

Fig. 7. Model of a PMP Resource Group [58].

6.1 Application of ACCESSORS in the PMP

From a modeling point of view, the fine-grained structure of ACCESSORS with its highly branched abstraction layers is necessary in order to gain a high expressiveness of the policy rules. However, from a implementation point of view, the number of utilized components should be kept low in order to reduce complexity. On that account, a mapping of the detailed ACCESSORS policy rules to similar PPM rules, is also recommended.

To that end, it is necessary to convert the access purposes specified by inquiring entities in ACCESSORS to Service Features. However, a Service Feature also defines certain permissions which are required in order to execute a particular code fragment. The focus on a broader range of possible attackers in ACCESSORS is not contradictory to the Service Features and a one-to-one mapping is possible without any further ado.

That is why the transition of data-centric targets modeled in ACCESSORS into PPM Resources poses the biggest problem for the mapping. In particular this implies that all Resources have to be replaced by new data-centric components. Nevertheless, ACCESSORS can be applied to the PMP, due to its modular architecture. In the PMP, each Resource Group is implemented as an independent functional unit which can be installed individually. Moreover, additional Resources can be added to a Resource Group at any point of time [51]. Therefore, existing Resources can be replaced by new data-centric ones in order to apply ACCESSORS to the PMP.

Figure 7 depicts the model of a Resource Group for Smart Watches. The Resource Group defines a common interface for all of its Resources. An arbitrary Resource, which provides the required functionality, can be plugged into the the Resource Group at runtime. That is, the Resources are concrete implementation artifacts of the interface for a given hardware (e. g., an Apple Watch or a Moto 360). The Resource Group also defines feasible Privacy Settings, i. e., how the user is able to restrict access to a particular Resource.

Figure 8 illustrates how this model has to be adapted in order to make the PMP compatible to ACCESSORS privacy rules. In the first instance, the hardware- or service-based focus of the Resource Groups has to be shifted to a data-centric one. The Resource Group given in the example deals with any kind

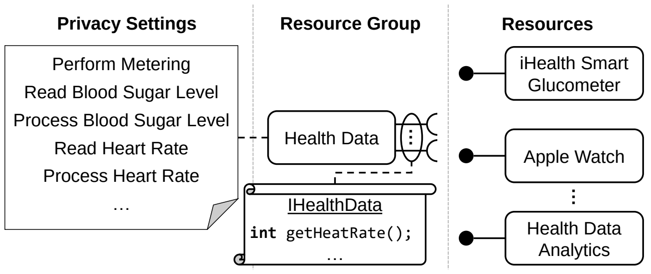

Fig. 8. Application of ACCESSORS in the PMP [58].

of health-related data. This data can be provided by legit health devices such as a glucometer or by novel Things such as a Smart Watch. Moreover, this Resource Group also deals with data consumers of health data such as analytics libraries. In order to be able to plug in all of these data processing units, the Resource Group's interface has to be broadened accordingly.

The PPM Resource Groups provide only a single plug for one Resource at a time. To support derivation transparency, i. e., to be able to model data which is assembled from various sources, the ACCESSORS Resource Groups need multiple plugs. For instance, it is possible to deduce the blood sugar level considerably accurate by monitoring the activities of a user and his or her eating behavior [69]. So, the Resource Group for health data has to be able to plug in a Resource capturing physical activities and a Resource gathering nutrition data, simultaneously. Furthermore, each Resource can be associated with multiple Resource Groups, e. g., a Smart Watch providing both, location and health data belongs to a location Resource Group as well as a health data Resource Group. The Privacy Settings for the new data-centric Resource Groups are carried over from the PPM's Resources that are pooled in the respective Resource Group.

That way, ACCESSORS can be mapped to the PPM. PPM rules are executable on the PMP and similar privacy systems. As the PMP runs on Android and Android is becoming increasingly pertinent to the IoT, such an implementation constitutes a serviceable privacy system for the IoT.

6.2 Application of ACCESSORS in PATRON

While the PMP provides access control to private data on Things such as Smartphones, the PATRON research project[5] adopts a different approach. As stream processing systems have proven to be a powerful means to process sensor information [13], they are of major importance for large-scale IoT apps for data processing in the Back-End Layer. For instance, the stream processing system of an IoT Smart Health app could process the users' heart rate, blood pressure, and GPS position to calculate their fitness levels or discover health problems. While many users want to benefit from such apps (e. g., to share their fitness data

[5] see http://patronresearch.de

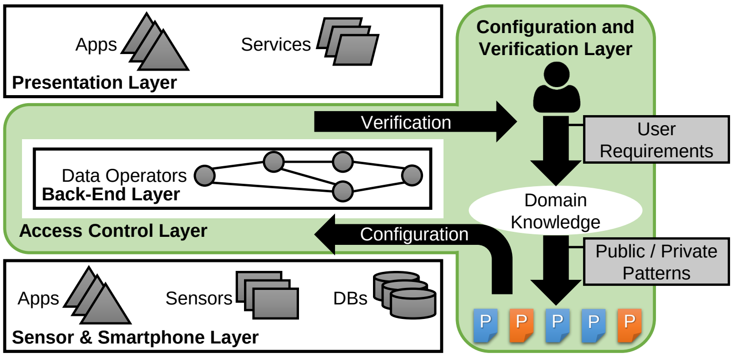

Fig. 9. Simplified PATRON Architecture [based on 54, 55].

with their health insurance in order to get a bonus), most users are afraid of the knowledge, which can be gained in addition (e. g., if a disease is discovered and the insurance fee rises). That is, users are not afraid of the data that is processed by an IoT app, but they want to conceal complex *patterns* (e. g., the disease in the example given above) within the data.

The blocking of certain attributes is therefore far too restrictive, as this also prevents the recognition of non-critical patterns (e. g., the fitness level). So, the user defines *private patterns*, i. e., patterns that have to be concealed, and *public patterns*, i. e., patterns that can be used in an IoT app, in PATRON. For the concealing of patterns, several techniques are available. PATRON selects the concealing technique which has the least negative impact on the quality of service.

Figure 9 shows the basic architecture of PATRON. In addition to the four layers of an IoT app (Sensor Layer, Smartphone Layer, Back-End Layer, and Presentation Layer), PATRON introduces two novel layers: the *Configuration and Verification Layer* and the *Access Control Layer*.

Configuration and Verification Layer. The Configuration and Verification Layer enables users to specify their requirements concerning privacy. This specification is made in natural language in order to also enable users who are not IT experts to express their requirements. For instance, a user might define the requirement "My insurance company must not be able to detect my unhealthy lifestyle based on the provided data.". Moreover, s/he can also express requirements towards the provided service of an IoT app such as "The insurance company has to be able to detect my fitness level.". It is obvious that users define these requirements, similar to ACCESSORS, based on a certain kind of data instead of sensors which provide this kind of data.

These requirements are transferred by domain experts into public and private patterns—the set of all these patterns is the configuration of PATRON. The assistance of domain experts is essential in this step, because their knowledge about applied analysis techniques and derivable knowledge is required. For instance, only medical experts know, which data sequence indicates a certain health problem. However, a physical attendance of the domain experts in the configuration process is not mandatory. Rather, it is sufficient if their knowledge is available in machine-processable form. Moreover, this accumulated knowledge base has to be expandable, as additional knowledge can be derived from existing data due to the introduction of novel sensors or new analysis techniques.

Besides the configuration of PATRON, the created patterns are also verified in this layer. To this end, all data which is forwarded from the Back-End Layer to the Presentation Layer (e. g., to an insurance company) is analyzed in the Configuration and Verification Layer. This enables to determine whether private patterns have been disclosed due to a misconfiguration or public patterns have been unnecessarily concealed. If this is the case, both, the patterns as well as the domain knowledge base can be adjusted correspondingly. Thus, the user has a confirmation that all of the privacy requirements are considered by PATRON.

Access Control Layer. The actual concealing is done in PATRON's Access Control Layer. The Access Control Layer encapsulate the Back-End-Layer completely, i. e., all incoming data can be analyzed by PATRON before it is shared with the IoT app's back-end and any results of the data operators within this layer can be restrained by PATRON. Various techniques are available for this purpose.

For instance, if the private pattern "eat sweets" (A) followed by "check blood sugar level" (B) followed by "inject insulin" (C) has to be concealed, PATRON could *suppress*, *obfuscate*, or *reorder* parts of the data. For example, this would have the following effect on the input stream $A \rightarrow B \rightarrow C \rightarrow D$: Suppression could simply drop any of the initial three events (A, B, or C), e. g., B resulting in the input stream $A \rightarrow C \rightarrow D$. Obfuscation modifies an event so that it seems to be a different event, e. g., event A could be disguised as event A' ("eat vegetables"). This leads to the input stream $A' \rightarrow B \rightarrow C \rightarrow D$. Finally, reordering could arrange event D at an earlier stage, e. g., between event A and B. This would also prevent the private pattern $A \rightarrow B \rightarrow C$, as the resulting input stream looks $A \rightarrow D \rightarrow B \rightarrow C$. For more information on the Access Control Layer, please refer to the respective literature [54, 55].

Privacy Requirements Elicitation via ACCESSORS. Even though there is system theoretical tool support for the semi-automatical translation of privacy requirements into public and private patterns, the elicitation of these requirements is highly complex [38]. As users should be able to formulate their requirements in natural language, these requirements have to be brought into a formalized, structured form before they can be processed.

The ACCESSORS basic model (see Fig. 5) can be used for this purpose. Guided by the three abstraction classes User Abstraction, Data Abstraction, and Context Abstraction, the user can define who (User Abstraction) should be able to access which patterns (Data Abstraction) in which situation (Context Abstraction). The Constraint Abstraction can be used to define whether it is a public or a private pattern. Moreover, the user can express the weight of each pattern, i. e., how important the respective pattern is for him or her. These weights are then considered by PATRON when selecting the concealing techniques.

Table 1. Comparison of Current Permission Models [based on 58].

Approach	**Data-Centric**	**Derivation Transparency**	**Extendable**	**Fine-Grained**	**Context-Sensitive**
DEFCON	✓	✗	✗	✗	✗
ACStream	✓	✗	✗	✗	✓
Android	✗	✗	(✓)	✗	✗
Selective Permissions	✗	✗	✗	✗	✗
CRêPE	✗	✗	✗	✗	✓
Apex	✗	✗	✗	(✓)	(✓)
YAASE	✗	✗	(✓)	✓	✗
Sorbet	✗	✗	✓	(✓)	(✓)
Retro-Skeleton	✗	(✓)	✓	(✓)	(✓)
Constroid	✓	✗	✗	(✓)	✓
AppGuard	✗	(✓)	✓	✓	(✓)
AFP	✗	✗	✓	✓	✓
DroidForce	✓	✗	✗	✓	✓
PMP	✗	✓	✓	✓	✓
ACCESSORS	✓	✓	✓	✓	✓

Yet, ACCESSORS fulfills in PATRON another substantial purpose. The Data Source Abstraction and Data Sink Abstraction can be used to model PATRON's domain knowledge base. Using these two modules, domain experts are able to specify which data (sources) can be used to derive certain information and how data sinks process data in a specific domain. The domain experts can create ACCESSORS rule fragments in which this expertise is made available. These fragments are then provided to users via the Data Abstraction. From the resulting ACCESSORS rules, the public and private patterns can be derived.

7 Discussion

ACCESSORS is fully **data-centered**, as all its protected entities (data producers as well as data consumers) are connected to a specific type of data (e. g., health data). Apps request access to this data without having to specify which sensor or system function provides this data. ACCESSORS thus also enables **derivation transparency**. Each protected data object can be provided by different sources. In addition, multiple sources can be combined to derive a specific type of data (e. g., the activity of a user can be derived from an accelerometer in combination with a position sensor). ACCESSORS makes it possible to model a single data source as producer of a variety of types of data. For instance, an Apple Watch provides both, location data and health data. The ACCESSORS model is **extendable**. On the one hand, additional data sources and sinks can be added at runtime to react to upcoming hardware. For example, the Apple Watch can measure blood sugar levels after a glucometer upgrade. On the other hand, ACCESSORS supports different types of entities. An app, a Smart Thing, or a user can all be specified as an inquiring entity. In this way, ACCESSORS is not limited to a fixed entity type. In the IoT context, where novel Things are released frequently, such an extensibility is indispensable. Furthermore, ACCESSORS is **fine-grained** and **context-sensitive**. This means that both, multi-valued constraints and spatial-temporal or situational conditions can be added to a policy rule.

Table 1 compares ACCESSORS with the permission models applied in the analyzed related work (see Sect. 4). In particular, the five key requirements towards a permission model for the IoT (see Sect. 3) are taken into account, which are [**R1**] data-centric policy rules, [**R2**] derivation transparency, [**R3**] extendable permission model, [**R4**] fine-grained policy rules, and [**R5**] context-sensitive policy rules. Due to the comprehensive abstraction approach covering users, data, data sinks, data sources, contexts, and constraints ACCESSORS is able to meet all requirements towards a permission model for the IoT.

8 Conclusion and Future Work

With the rise of the IoT, there constantly arise novel application fields for this technology. The IoT can improve the quality of life and living (Smart Homes), facilitate the treatment of chronic diseases (Smart Health), and make road traffic safer and more comfortable (Smart Cars), just to name a few of such application fields. For this purpose, however, a great amount of private data about the user has to be collected. Therefore, such applications not only improve the quality of life, but also pose a threat towards privacy. Thus, users need powerful, yet easy to manage mechanisms to control the access to their data.

In this paper we examine whether the currently existing privacy systems for Things are also suitable for the IoT. Since the analysis of these systems shows that the permission models applied by them do not meet all the requirements towards a privacy system for the IoT—namely, data-centric policy rules, derivation transparency, extendable permission model, fine-grained policy rules, and context-sensitive policy rules—we come up with a novel permission model called ACCESSORS. We show that this model not only meets all requirements towards a permission model for the IoT, but that it can also be easily integrated into existing privacy systems for the IoT. We illustrate this exemplarily for the PMP, a privacy system for the Smartphone Layer, and for PATRON, aprivacy systems for the Back-End Layer.

As shown in Sect. 6, ACCESSORS can be used in both, privacy systems for the Smartphone Layer and privacy systems for the Back-End Layer. The aim of future work therefore is to combine these two types of privacy system via ACCESSORS. Since both types can be configured by using ACCESSORS, users would be able to make their privacy requirements elicitation once and then transfer it to all of their Things as well as the back-end. Synergy effects can be achieved by this combination. On the one hand, particularly confidential data can be blocked at an early stage in the Smartphone Layer which increases data security, as this data thereby never leaves the user's Thing. On the other hand, a pattern-based privacy solution in the back-end enables the highest possible quality of service, since certain attributes are not systematically filtered out. Instead, only complex sequences of attributes declared as private patterns are concealed, while public attribute sequences remain unaffected.

Alpers et al. [3] describe an approach how privacy rules can be defined and managed at a central site once and then transferred and applied to any end device of a user. Future work will have to consider how this approach can be applied to ACCESSORS. In addition, it has to be assessed to what extent the approach can be extended so that it can also be embedded in the Configuration and Verification Layer of PATRON.

Acknowledgments This paper is part of the PATRON research project which is commissioned by the Baden-Württemberg Stiftung gGmbH. The authors would like to thank the BW-Stiftung for the funding of this research.

References

1. Aggarwal, C.C., Ashish, N., Sheth, A.: The Internet of Things: A Survey from the Data-Centric Perspective. In: Aggarwal, C.C. (ed.) Managing and Mining Sensor Data, chap. 12, pp. 383–428. Springer (2013). https://doi.org/10.1007/978-1-4614-6309-2_12
2. Agrawal, D., El Abbadi, A., Wang, S.: Secure and Privacy-preserving Data Services in the Cloud: A Data Centric View. Proceedings of the VLDB Endowment **5**(12), 2028–2029 (2012). https://doi.org/10.14778/2367502.2367569
3. Alpers, S., Oberweis, A., Pieper, M., Betz, S., Fritsch, A., Schiefer, G., Wagner, M.: PRIVACY-AVARE: An Approach to Manage and Distribute Privacy Settings. In: Proceedings of the 2017 3rd IEEE International Conference on Computer and Communications. pp. 1460–1468. ICCC '17 (2017). https://doi.org/10.1145/3098279.3098535
4. Aman, M.N., Chua, K.C., Sikdar, B.: Secure Data Provenance for the Internet of Things. In: Proceedings of the 3rd ACM International Workshop on IoT Privacy, Trust, and Security. pp. 11–14. IoTPTS '17 (2017). https://doi.org/10.1145/3055245.3055255
5. Backes, M., Gerling, S., Hammer, C., Maffei, M., von Styp-Rekowsky, P.: AppGuard: Enforcing User Requirements on Android Apps. In: Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 543–548. TACAS '13 (2013). https://doi.org/10.1007/978-3-642-36742-7_39
6. Backes, M., Gerling, S., Hammer, C., Maffei, M., Styp-Rekowsky, P.: AppGuard – Fine-Grained Policy Enforcement for Untrusted Android Applications. In: Garcia-Alfaro, J., Lioudakis, G., Cuppens-Boulahia, N., Foley, S., Fitzgerald, W.M. (eds.) Data Privacy Management and Autonomous Spontaneous Security: 8th International Workshop, DPM 2013, and 6th International Workshop, SETOP 2013, Egham,

UK, September 12-13, 2013, Revised Selected Papers, pp. 213–231. Springer (2014). https://doi.org/10.1007/978-3-642-54568-9_14

7. Barrera, D., Kayacik, H.G., van Oorschot, P.C., Somayaji, A.: A Methodology for Empirical Analysis of Permission-based Security Models and Its Application to Android. In: Proceedings of the 17th ACM Conference on Computer and Communications Security. pp. 73–84. CCS '10 (2010). https://doi.org/10.1145/1866307.1866317
8. Bitsaki, M., Koutras, C., Koutras, G., Leymann, F., Mitschang, B., Nikolaou, C., Siafakas, N., Strauch, S., Tzanakis, N., Wieland, M.: An Integrated mHealth Solution for Enhancing Patients' Health Online. In: Proceedings of the 6th European Conference of the International Federation for Medical and Biological Engineering. pp. 695–698. MBEC '14 (2014)
9. Brush, A.B., Lee, B., Mahajan, R., Agarwal, S., Saroiu, S., Dixon, C.: Home Automation in the Wild: Challenges and Opportunities. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2115–2124. CHI '11 (2011). https://doi.org/10.1145/1978942.1979249
10. Cao, J., Carminati, B., Ferrari, E., Tan, K.L.: ACStream: Enforcing Access Control over Data Streams. In: Proceedings of the 2009 IEEE 25th International Conference on Data Engineering. pp. 1495–1498. ICDE '09 (2009)
11. Chin, E., Felt, A.P., Sekar, V., Wagner, D.: Measuring User Confidence in Smartphone Security and Privacy. In: Proceedings of the Eighth Symposium on Usable Privacy and Security. pp. 1:1–1:16. SOUPS '12 (2012). https://doi.org/10.1145/2335356.2335358
12. Conti, M., Nguyen, V.T.N., Crispo, B.: CRêPE: Context-related Policy Enforcement for Android. In: Burmester, M., Tsudik, G., Magliveras, S., Ilić, I. (eds.) Information Security: 13th International Conference, ISC 2010, Boca Raton, FL, USA, October 25-28, 2010, Revised Selected Papers, pp. 331–345. Springer (2011). https://doi.org/10.1007/978-3-642-18178-8_29
13. Cugola, G., Margara, A.: Processing Flows of Information: From Data Stream to Complex Event Processing. ACM Computing Surveys **44**(3), 15:1–15:62 (2012). https://doi.org/10.1145/2187671.2187677
14. Davies, N., Taft, N., Satyanarayanan, M., Clinch, S., Amos, B.: Privacy Mediators: Helping IoT Cross the Chasm. In: Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications. pp. 39–44. HotMobile '16 (2016). https://doi.org/10.1145/2873587.2873600
15. Davis, B., Chen, H.: RetroSkeleton: Retrofitting Android Apps. In: Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services. pp. 181–192. MobiSys '13 (2013). https://doi.org/10.1145/2462456.2464462
16. Davis, B., Sanders, B., Khodaverdian, A., Chen, H.: I-ARM-Droid: A Rewriting Framework for In-App Reference Monitors for Android Applications. In: Proceedings of the 2012 IEEE Conference on Mobile Security Technologies. pp. 28:1–28:9. MoST '12 (2012)
17. Dey, A.K.: Understanding and Using Context. Personal and Ubiquitous Computing **5**(1), 4–7 (2001). https://doi.org/10.1007/s007790170019
18. Enck, W., Ongtang, M., McDaniel, P.: Understanding Android Security. IEEE Security and Privacy **7**(1), 50–57 (2009)
19. Felt, A.P., Egelman, S., Wagner, D.: I've Got 99 Problems, but Vibration Ain't One: A Survey of Smartphone Users' Concerns. In: Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. pp. 33–44. SPSM '12 (2012). https://doi.org/10.1145/2381934.2381943
20. Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D.: Android Permissions: User Attention, Comprehension, and Behavior. In: Proceedings of the Eighth

Symposium on Usable Privacy and Security. pp. 3:1–3:14. SOUPS '12 (2012). https://doi.org/10.1145/2335356.2335360
21. Fragkaki, E., Bauer, L., Jia, L., Swasey, D.: Modeling and Enhancing Android's Permission System. In: Foresti, S., Yung, M., Martinelli, F. (eds.) Computer Security – ESORICS 2012: 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings, pp. 1–18. Springer (2012). https://doi.org/10.1007/978-3-642-33167-1_1
22. Google Inc.: Android Things. https://developer.android.com/things (May 2018)
23. Hamlen, K.W., Jones, M.: Aspect-oriented In-lined Reference Monitors. In: Proceedings of the Third ACM SIGPLAN Workshop on Programming Languages and Analysis for Security. pp. 11–20. PLAS '08 (2008). https://doi.org/10.1145/1375696.1375699
24. Harle, R.K., Tailor, S., Zidek, A.: Bellrock – Anonymous Proximity Beacons From Personal Devices. In: Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications. pp. 284–293. PerCom '18 (2018)
25. He, Y., Barman, S., Wang, D., Naughton, J.F.: On the Complexity of Privacy-preserving Complex Event Processing. In: Proceedings of the Thirtieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. pp. 165–174. PODS '11 (2011). https://doi.org/10.1145/1989284.1989304
26. Henrik, Z.J., Garcia, M.O., Klaus, W.: Privacy in the Internet of Things: threats and challenges. Security and Communication Networks **7**(12), 2728–2742 (2014). https://doi.org/10.1002/sec.795
27. Hilty, M., Pretschner, A., Basin, D., Schaefer, C., Walter, T.: A Policy Language for Distributed Usage Control. In: Biskup, J., López, J. (eds.) Computer Security – ESORICS 2007: 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24 — 26, 2007. Proceedings, pp. 531–546. Springer (2007). https://doi.org/10.1007/978-3-540-74835-9_35
28. Istepanian, R.S.H., Hu, S., Philip, N., Sungoor, A.: The Potential of Internet of m-health Things "m-IoT" for Non-Invasive Glucose Level Sensing. In: Proceedings of the 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. pp. 5264–5266. EMBS '11 (2011). https://doi.org/10.1109/IEMBS.2011.6091302
29. Jordan, M., Mitchell, T.: Machine learning: Trends, perspectives, and prospects. Science **349**(6245), 255–260 (2015). https://doi.org/10.1126/science.aaa8415
30. Kalkov, I., Franke, D., Schommer, J.F., Kowalewski, S.: A Real-time Extension to the Android Platform. In: Proceedings of the 10th International Workshop on Java Technologies for Real-time and Embedded Systems. pp. 105–114. JTRES '12 (2012). https://doi.org/10.1145/2388936.2388955
31. Khan, R., Khan, S.U., Zaheer, R., Khan, S.: Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges. In: Proceedings of the 2012 10th International Conference on Frontiers of Information Technology. pp. 257–260. FIT '12 (2012). https://doi.org/10.1109/FIT.2012.53
32. Knöll, M.: Diabetes City: How Urban Game Design Strategies Can Help Diabetics. In: Weerasinghe, D. (ed.) Electronic Healthcare: First International Conference, eHealth 2008, London, UK, September 8-9, 2008. Revised Selected Papers, pp. 200–204. Springer (2009). https://doi.org/10.1007/978-3-642-00413-1_28
33. Knöll, M.: "On the Top of High Towers . . . " Discussing Locations in a Mobile Health Game for Diabetics. In: Proceedings of the 2010 IADIS International Conference Game and Entertainment Technologies. pp. 61–68. MCCSIS '10 (2010)
34. Kovatchev, B.P., Gonder-Frederick, L.A., Cox, D.J., Clarke, W.L.: Evaluating the Accuracy of Continuous Glucose-Monitoring Sensors. Diabetes Care **27**(8), 1922–1928 (2004). https://doi.org/10.2337/diacare.27.8.1922

35. Kozlov, D., Veijalainen, J., Ali, Y.: Security and Privacy Threats in IoT Architectures. In: Proceedings of the 7th International Conference on Body Area Networks. pp. 256–262. BodyNets '12 (2012). https://doi.org/10.1145/3098279.3098535
36. Metzger, A., Cassales Marquezan, C.: Future Internet Apps: The Next Wave of Adaptive Service-Oriented Systems? In: Abramowicz, W., Llorente, I.M., Surridge, M., Zisman, A., Vayssière, J. (eds.) Towards a Service-Based Internet, pp. 230–241. Springer (2011). https://doi.org/10.1007/978-3-642-24755-2_22
37. Migliavacca, M., Papagiannis, I., Eyers, D.M., Shand, B., Bacon, J., Pietzuch, P.: DEFCON: High-performance Event Processing with Information Security. In: Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference. pp. 1–15. USENIXATC '10 (2010)
38. Mindermann, K., Riedel, F., Abdulkhaleq, A., Stach, C., Wagner, S.: Exploratory Study of the Privacy Extension for System Theoretic Process Analysis (STPA-Priv) to Elicit Privacy Risks in eHealth. In: Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference Workshops. pp. 90–96. REW '17 (2017). https://doi.org/10.1109/REW.2017.30
39. Nauman, M., Khan, S., Zhang, X.: Apex: Extending Android Permission Model and Enforcement with User-defined Runtime Constraints. In: Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security. pp. 328–332. ASIACCS '10 (2010). https://doi.org/10.1145/1755688.1755732
40. Park, J., Sandhu, R.: The $\mathrm{UCON}_{\mathrm{ABC}}$ Usage Control Model. ACM Transactions on Information and System Security **7**(1), 128–174 (2004). https://doi.org/10.1145/984334.984339
41. Perera, C., Zaslavsky, A., Christen, P.: Context Aware Computing for The Internet of Things: A Survey. IEEE Communications Surveys & Tutorials **16**(1), 414–454 (2014). https://doi.org/10.1109/SURV.2013.042313.00197
42. Rasthofer, S., Arzt, S., Lovat, E., Bodden, E.: DroidForce: Enforcing Complex, Data-centric, System-wide Policies in Android. In: Proceedings of the 2014 Ninth International Conference on Availability, Reliability and Security. pp. 40–49. ARES '14 (2014). https://doi.org/10.1109/ARES.2014.13
43. Russello, G., Crispo, B., Fernandes, E., Zhauniarovich, Y.: YAASE: Yet Another Android Security Extension. In: Proceeding of the 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing. pp. 1033–1040. PASSAT '11 (2011). https://doi.org/10.1109/PASSAT/SocialCom.2011.151
44. Sarkar, S., Misra, S.: Theoretical modelling of fog computing: a green computing paradigm to support IoT applications. IET Networks **5**(2), 23–29 (2016). https://doi.org/10.1049/iet-net.2015.0034
45. Schreckling, D., Posegga, J., Hausknecht, D.: Constroid: Data-centric Access Control for Android. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing. pp. 1478–1485. SAC '12 (2012). https://doi.org/10.1145/2245276.2232012
46. Scoccia, G.L., Malavolta, I., Autili, M., Di Salle, A., Inverardi, P.: User-centric Android Flexible Permissions. In: Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion. pp. 365–367. ICSE-C '17 (2017). https://doi.org/10.1109/ICSE-C.2017.84
47. Sekar, L.P., Gankidi, V.R., Subramanian, S.: Avoidance of Security Breach Through Selective Permissions in Android Operating System. ACM SIGSOFT Software Engineering Notes **5**(37), 1–9 (2012). https://doi.org/10.1145/2347696.2347711
48. Sellwood, J., Crampton, J.: Sleeping Android: The Danger of Dormant Permissions. In: Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices. pp. 55–66. SPSM '13 (2013). https://doi.org/10.1145/2516760.2516774

49. Sicari, S., Rizzardi, A., Grieco, L.A., Coen-Porisini, A.: Security, privacy and trust in Internet of Things: The road ahead. Computer Networks **76**, 146–164 (2015). https://doi.org/10.1016/j.comnet.2014.11.008
50. Siewiorek, D.: Generation Smartphone. IEEE Spectrum **49**(9), 54–58 (2012). https://doi.org/10.1109/MSPEC.2012.6281134
51. Stach, C.: How to Assure Privacy on Android Phones and Devices? In: Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management. pp. 350–352. MDM '13 (2013). https://doi.org/10.1109/MDM.2013.54
52. Stach, C.: Secure Candy Castle — A Prototype for Privacy-Aware mHealth Apps. In: Proceedings of the 2016 IEEE 17th International Conference on Mobile Data Management. pp. 361–364. MDM '16 (2016). https://doi.org/10.1109/MDM.2016.64
53. Stach, C., Alpers, S., Betz, S., Dürr, F., Fritsch, A., Mindermann, K., Palanisamy, S.M., Schiefer, G., Wagner, M., Mitschang, B., Oberweis, A., Wagner, S.: The AVARE PATRON: A Holistic Privacy Approach for the Internet of Things. In: Proceedings of the 15th International Conference on Security and Cryptography. pp. 1–6. SECRYPT '18 (2018)
54. Stach, C., Dürr, F., Mindermann, K., Palanisamy, S.M., Tariq, M.A., Mitschang, B., Wagner, S.: PATRON — Datenschutz in Datenstromverarbeitungssystemen. In: Informatik 2017: Digitale Kulturen, Tagungsband der 47. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 25.9-29.9.2017, Chemnitz. LNI, vol. 275, pp. 1085–1096 (2017), *(in German)*
55. Stach, C., Dürr, F., Mindermann, K., Palanisamy, S.M., Wagner, S.: How a Pattern-based Privacy System Contributes to Improve Context Recognition. In: Proceedings of the 2018 IEEE International Conference on Pervasive Computing and Communications Workshops. pp. 238–243. CoMoRea '18 (2018)
56. Stach, C., Mitschang, B.: Privacy Management for Mobile Platforms – A Review of Concepts and Approaches. In: Proceedings of the 2013 IEEE 14th International Conference on Mobile Data Management. pp. 305–313. MDM '13 (2013). https://doi.org/10.1109/MDM.2013.45
57. Stach, C., Mitschang, B.: Design and Implementation of the Privacy Management Platform. In: Proceedings of the 2014 IEEE 15th International Conference on Mobile Data Management. pp. 69–72. MDM '14 (2014). https://doi.org/10.1109/MDM.2014.14
58. Stach, C., Mitschang, B.: ACCESSORS: A Data-Centric Permission Model for the Internet of Things. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy. pp. 30–40. ICISSP '18 (2018). https://doi.org/10.5220/0006572100300040
59. Stach, C., Schlindwein, L.F.M.: Candy Castle — A Prototype for Pervasive Health Games. In: Proceedings of the 2012 IEEE International Conference on Pervasive Computing and Communications Workshops. pp. 501–503. PerCom '12 (2012). https://doi.org/10.1109/PerComW.2012.6197547
60. Stach, C., Steimle, F., Mitschang, B.: The Privacy Management Platform: An Enabler for Device Interoperability and Information Security in mHealth Applications. In: Proceedings of the 11th International Conference on Health Informatics. pp. 27–38. HEALTHINF '18 (2018). https://doi.org/10.5220/0006537300270038
61. Stach, C., Steimle, F., Franco da Silva, A.C.: TIROL: The Extensible Interconnectivity Layer for mHealth Applications. In: Damaševičius, R., Mikašytė, V. (eds.) Information and Software Technologies: 23nd International Conference, ICIST 2017, Druskininkai, Lithuania, October 12-14, 2017, Proceedings, pp. 190–202. Springer (2017). https://doi.org/10.1007/978-3-319-67642-5_16

62. Svangren, M.K., Skov, M.B., Kjeldskov, J.: The Connected Car: An Empirical Study of Electric Cars As Mobile Digital Devices. In: Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services. pp. 6:1–6:12. MobileHCI '17 (2017). https://doi.org/10.1145/3098279.3098535
63. Takabi, H., Joshi, J.B.D., Ahn, G.J.: Security and Privacy Challenges in Cloud Computing Environments. IEEE Security and Privacy **8**(6), 24–31 (2010). https://doi.org/10.1109/MSP.2010.186
64. The European Parliament and the Council of the European Union: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Official journal of the european union, European Union (2016)
65. Vashist, S.K., Schneider, E.M., Luong, J.H.: Commercial Smartphone-Based Devices and Smart Applications for Personalized Healthcare Monitoring and Management. Diagnostics **4**(3), 104–128 (2014). https://doi.org/10.3390/diagnostics4030104
66. Wei, X., Gomez, L., Neamtiu, I., Faloutsos, M.: Permission Evolution in the Android Ecosystem. In: Proceedings of the 28th Annual Computer Security Applications Conference. pp. 31–40. ACSAC '12 (2012). https://doi.org/10.1145/2420950.2420956
67. Weiser, M.: The Computer for the 21st Century. Scientific American **265**(3), 94–105 (1991). https://doi.org/10.1109/MSPEC.2012.6281134
68. Xie, X., Ray, I., Adaikkalavan, R., Gamble, R.: Information Flow Control for Stream Processing in Clouds. In: Proceedings of the 18th ACM Symposium on Access Control Models and Technologies. pp. 89–100. SACMAT '13 (2013). https://doi.org/10.1145/2462410.2463205
69. Zeevi, D., Korem, T., Zmora, N., Israeli, D., Rothschild, D., Weinberger, A., Ben-Yacov, O., Lador, D., Avnit-Sagi, T., Lotan-Pompan, M., Suez, J., Mahdi, J.A., Matot, E., Malka, G., Kosower, N., Rein, M., Zilberman-Schapira, G., Dohnalová, L., Pevsner-Fischer, M., Bikovsky, R., Halpern, Z., Elinav, E., Segal, E.: Personalized Nutrition by Prediction of Glycemic Responses. Cell **163**(5), 1079–1094 (2015). https://doi.org/10.1016/j.cell.2015.11.001

All links were last followed on June 1, 2018.