Data Protection in a Cloud-enabled Smart Grid[⋆]

Alexander Fromm[1], Florian Kelbert[2], and Alexander Pretschner[2]

[1] Karlsruhe Institute of Technology
fromm@kit.edu
[2] Technische Universität München
kelbert@in.tum.de, pretschn@in.tum.de

Abstract. Today's electricity grid is evolving into the *smart grid* which ought to be reliable, flexible, efficient, and sustainable. To fulfill these requirements, the smart grid draws on a plenty of core technologies. Advanced Metering Infrastructure (AMI). These technologies facilitate easy and fast accumulation of different data, e.g. fine-grained meter readings. Various security and privacy concerns w.r.t. the gathered data arise, since research has shown that it is possible to deduce and extract user behaviour from smart meter readings. Hence, these meter readings are very sensitive and require appropriate protection.
Unlike other data protection approaches that are primarily based on data obfuscation and data encryption, we introduce a usage control based data protection mechanism for the smart grid. We show how the concept of distributed data usage control can be integrated with smart grid services and concretize this approach for an energy marketplace that runs on a cloud platform for performance, scalability, and economic reasons.

1 Introduction

The smart grid is regarded as the next-generation electricity grid and is expected to address the deficits and problems of the current power grid [8]. It encompasses an intelligent networking and management of power generators, power consumers, and power storage in the energy transmission and distribution network. For doing so, the smart grid draws on various technologies [18], including the Advanced Metering Infrastructure (AMI) [17] which is fundamental for real-time measurements and time-of-use meters. By means of smart meter devices consumers' energy consumption data can be observed and collected at highly frequent time periods, e.g. each second. Such fine-grained meter readings enable dynamic pricing and allow to forecast energy demand [23]. Moreover, utility providers can facilitate load balancing, peak load reduction, and more efficient network management.

However, with the advent of the smart grid and its pervasiveness in our daily lives, new challenges and concerns arise—in particular about energy prosumers'

[⋆] Work supported by the "Peer Energy Cloud" project funded by the German Federal Ministry of Economics and Technology as well as by the Google focused research award "Usage control in the cloud."

privacy [5, 6], a prosumer being an energy producer and consumer at the same time. Research has shown that it is possible to deduce and infer private and intimate details of prosumers' daily lives from fine-grained meter readings [16]: data mining technologies allow for the extraction of residents' lifestyles including breakfast, lunch, and dinner activities or wake, presence, absence, and sleep cycles. According to [4], there will be the temptation to sell such information, e.g. energy usage or appliance data, either in identifiable customer level, anonymized, or in aggregated form to third parties, e.g.marketers seeking commercial gain. Thus, protecting prosumers' smart meter readings is a particular challenge in the smart grid, especially once meter readings are released to third parties.

To tackle this problem, we introduce usage control concepts [19, 20] to the smart grid domain. In contrast to other data protection mechanisms, like access control that monitors and controls who can access and interact with sensitive data, usage control is concerned with how data may or may not be used once (initial) access to it *has been* granted. Data usage requirements, like "do not distribute my meter readings" are specified in usage control policies[3] and are enforced by the usage control infrastructure.

In this paper we integrate usage control for technical reasons (we explain them later) into the cloud PaaS[4] model in order to provide a secure platform for operating arbitrary smart grid services. The PaaS level also specifies the system boundaries for our usage control mechanisms. We refer to a smart grid energy marketplace as one specific smart grid service example. Such an energy marketplace will be an essential component of the future smart grid, since its very idea is to facilitate the trading of energy [2,3]. This is because governments currently reduce their subventions for renewable energy (in Germany there exist plans to abolish renewable energy funding [27]), which as a side effect releases utility providers from the burden of having to buy prosumers' excess energy. Hence, prosumers must either consume the energy themselves or sell it to other consumers, thus laying the grounds for an open energy marketplace.

For a marketplace provider, or more generally speaking for arbitrary smart grid service providers, it is uncertain whether their services will be accepted and used by prosumers. Hence, there is a considerable risk to lose significant portions of initial capital investments. To mitigate this risk, it is beneficial to run these services on a cloud computing infrastructure and exploit the opportunity to adapt computation and storage resources on demand (*scalability*) when the prosumers' service usage increases or decreases. This way, service providers can minimize their investments and pay for consumed resources only. Moreover, service providers can leverage the broadband network access and the implicit data back up and replication mechanisms. Such functionalities are very useful because an AMI system with 2 million consumers and a data rate of 1KB per minute and per smart meter produces 3TB of data per day [25].

[3] Usage control policies could also comprise cloud privacy requirements required by governmental regulations.

[4] *Platform as a Service (PaaS)* [15] is a cloud computing service model which provides a software platform for development and deployment of cloud applications (services).

Example attack scenario. Assume that Alice has registered her smart meter device with a smart grid marketplace, which runs as a smart grid service on a cloud PaaS, in order to buy and sell energy. Therefore, she serves the marketplace with her smart meter readings. After a while she starts to receive unwanted, personalized advertisement mails. Based on this scenario, at least two attacker models are conceivable: a malicious marketplace employee, e.g. an administrator, and a malicious marketplace provider. In the former case a malicious marketplace administrator would copy (e.g. by sending smart meter readings to a remote server) and sell smart meter readings to the advertising industry. In the latter case, the marketplace service is malicious from the outset and collects the meter values not only for trading purposes but also for the generation of user behaviour profiles which are then sold to the advertising industry.
Problem. In this paper we tackle the problem of protecting sensitive smart meter readings in the smart grid and focus on security and privacy aspects once meter readings are released to untrusted smart grid services. More generally, we tackle the problem how to protect sensitive data once those are transmitted to untrusted (cloud) smart grid services.
Solution. We introduce a data protection mechanism, namely usage control, for smart grid services and integrate our usage control concept in a cloud PaaS infrastructure. Our solution allows to release prosumers' smart meter readings *without loss of control over their further usage.*
Contribution. We are not aware of any concepts or systems in the smart grid domain that protect meter readings on the basis of specified policies even after they have been released. We contribute our data protection solution for cloud-enabled smart grid services.
Organization. We organize our work as follows: §2 illustrates the setting with the scenario where our usage control mechanisms are deployed. §3 describes the main architecture and design of our cloud-enabled solution. §4 analyzes our solution w.r.t. weaknesses and limitations. §5 describes related work on data protection in the smart grid and section §6 summarizes our results and gives a perspective on future work.

2 Setting

The smart grid addresses one major challenge that is accompanied with distributed and renewable energy generation: time of energy generation can hardly be influenced and will most of the time not meet energy demand. Since centralized capacities for saving energy (e.g. pumped-storage hydroelectricity) are limited, one idea is to align energy demand with energy generation. To this end, web-based energy marketplaces are being developed [2, 3], aiming at a price-driven balance of energy supply and demand. Such marketplaces bring together producers and consumers of energy, allow for negotiation of flexible energy contracts, with the goal to decentralize the problem of energy over- and underproduction. In order to automatize trading and real-time pricing, prosumers must provide their preferences and smart meter readings to the marketplace. For instance, in Fig. 1 such meter readings originate from a smart meter that collects the consumed and produced values via a *Home Area Network (HAN)* from the local energy consuming (stove, fridge), producing (solar panel), and storing (electric car) appliances. The collected meter readings are transmitted periodically (e.g. each second) to the marketplace where these data are continuously analyzed and aggregated to supply and demand packages. Based on these packages, marketplace participants have buy and sell options in accordance with their energy demand, i.e. they can specify when, how much, and at what price they want to buy or sell energy. Furthermore, the marketplace functionality can be extended by external services (cf. service S2 in Fig. 1), e.g. agents for automatic energy trading on users' behalf.

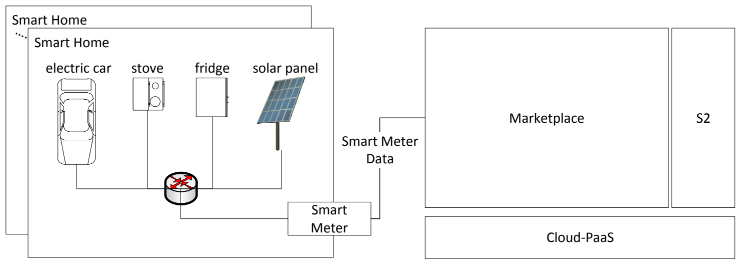

Fig. 1. Setting at a glance.

Considering such a marketplace concept and the fact that a utility provider has potentially millions of smart meters[5], it is no surprise that such a marketplace puts high requirements on the underlying infrastructure—in particular in terms of broadband network access, performance, scalability, and flexibility. In order to meet these requirements, it is reasonable to deploy the marketplace on a cloud platform (cf. *Peer Energy Cloud* project [2]), which provides the needed capacities for arbitrary services on demand.

On the one hand the use of cloud computing brings along several benefits like on-demand self-service, resource pooling, elasticity, and measured services (i.e. use of a cloud service on pay-per-use or charge-per-use basis) [9,15]. On the other hand cloud computing raises security and privacy issues w.r.t. protecting the smart meter data from unauthorized usage: once sensitive data leave the user-controlled smart meter device, controlling its further usage is not possible. For instance, this happens when the smart meter transmits meter readings to the marketplace (cf. Fig. 1). At this point the marketplace provider has full access to the received meter data and is able to e.g. create user behaviour profiles [16]. To tackle this problem, **we present a solution based on data usage control** that we deploy at the cloud PaaS service model. In the next section, we describe in detail our usage control infrastructure, its components, and its integration into the PaaS service layer.

[5] The *Los Angeles Department of Water and Power* has 2.1 million consumer accounts that correspond to approximately 2.1 million smart meters [25].

3 Usage Control for the Smart Grid

In a computing system, data usually exists in different representations at different abstraction layers [22]: smart meter readings can be represented as a file (file system), as a database record (database), or as a Java object (Java Virtual Machine). In order to enforce usage control policies and to track the flow of data across different abstraction layers, it is necessary to integrate usage control components and mechanisms at each of these layers. For doing so, our usage control architecture[6] (cf. Fig. 2) consists of a *Policy Enforcement Point (PEP)*, a *Policy Decision Point (PDP)*, a *Policy Information Point (PIP)*, and a *Policy Management Point (PMP)*[7].

In the following section, we give a conceptual view on our usage control architecture and describe an integration of their components at one abstraction layer, namely at the Java layer.

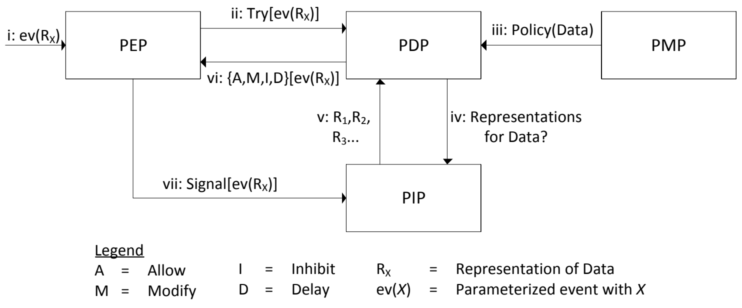

Fig. 2. Usage control architecture and components.

3.1 Architecture

The PEP is the monitoring and enforcement entity and is closely tied to one abstraction layer. Therefore, the PEP is able to observe, intercept, and forward usage control relevant events to the PDP (cf. *ii* + *vi*, Fig. 2) which then takes decisions (allow, delay, forbid, or modify an event [21]) on the basis of usage control policies provided by the PMP (cf. *iii*, Fig. 2) as well as the data flow state provided by the PIP (cf. *iv* + *v*, Fig. 2). The PEP enforces the decision and signals the intercepted events for data tracking purposes to the PIP (cf. *vii*, Fig. 2). As the PIP tracks the data flow across different abstraction layers, it knows at a specific moment in time which data is in which representation at which abstraction layer [22]. If a relevant data flow event occurs on any abstraction layer, the PIP receives this information from the corresponding PEP (cf. *vii*, Fig.

[6] In contrast to other approaches like *Digital Rights Management*, usage control is not tailored for one specific purpose, but is open for any field of application, including privacy protection [19].

[7] Demos are available online at https://www22.cs.tum.edu/index.php?id=64

2) and updates the data flow state of the system. The PMP stores, translates, instantiates, and delivers policies on demand.

3.2 Integration

We integrate the usage control components (cf. §3.1) at one specific abstraction layer that is used for smart grid services running on a cloud PaaS platform. First, we outline the reasons for choosing the cloud PaaS service model.

The cloud computing paradigm provides different levels of abstraction for service deployment: *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)*, and *Software as a Service (SaaS)* [15]. Depending on the deployed service model, the integration of usage control comes with various pros and cons. For instance, the integration at the IaaS level would make circumventing our usage control infrastructure more difficult than at the SaaS level. However, the IaaS level provides less context information about overlying services, hence making it more difficult to know the semantics of system events. In turn, integrating usage control at the SaaS level comes with the problem that each SaaS service must implement usage control mechanisms itself. In practice, it is unfeasible to ensure that each service implements these mechanisms correctly, since large amounts of such services could be provided by different service providers. To mitigate these obstacles, our solution provides usage control mechanisms at the PaaS level (cf. Fig. 1), assuming that the PaaS provider is trustworthy[8] and does not attack the usage control infrastructure. This way, events originating from software services running on the PaaS level can be intercepted and evaluated without interference with the underlying infrastructure.

For proof-of-concept, we provide a usage control augmented Java runtime environment on the PaaS level for operating arbitrary java-based smart grid services[9], e.g. an energy marketplace as described in §2. We implement the PEP as a *Java programming language agent* (using *java.lang.instrument* package) into the *Java Virtual Machine (JVM)*. By using the *Byte Code Engineering Library*, the PEP fetches, (re)transforms, and instruments the Java bytecode in order to extract and transfer context information from the running smart grid service to the PDP and PIP (cf. §3.1). More precisely, the PEP instruments every *load*, *store*, *put*, *get*, and *invoke* bytecode instruction with method invocations that reveal the occurrence of local variable assignments, access to class attributes, and method invocations within a java-based smart grid service. Moreover, the PEP allows, modifies, or inhibits the execution of Java (system-)class methods depending on the PDP's policy evaluation result. This way, our PEP is able to enforce usage control policies within smart grid services and to reveal context information for (explicit) data flow tracking purposes to the PIP. In the following, we describe how our PEP component instruments Java bytecode.

[8] How trustworthiness can technically be achieved is discussed in [13].

[9] A plenty of frameworks and tools exist that facilitate the application development in Java. Therefore, we assume that Java is the first choice for the development of smart grid services, especially for a smart grid energy marketplace [2].

To illustrate our approach, we consider the Java example code in Listing 1.1 that is deployed within the energy marketplace (cf. §1) for transmitting smart meter data; we assume that the implementations of *readMr()* and *sendData()* use Java system-classes in order to read data from the local file system and to send data via the network, respectively.

```
int meter_reading, b;
meter_reading = readMr();
b = meter_reading;
sendData(b);
```

Listing 1.1. Java source: send meter readings.

```
0: invokestatic   #16..
        ..//Method readMr:()I
3: istore_1
4: iload_1
5: istore_2
6: iload_2
7: invokestatic   #20..
      ..//Method sendData:(I)V
```

Listing 1.2. Bytecode of Listing 1.1.

An attacker may misuse the code in Listing 1.1 to transfer not permitted copies of meter readings to remote systems that are under his control (cf. attack scenario in §1). In order to thwart such attacks, the PEP must enforce the (high-level) usage control policy *"do not distribute my meter readings"* (P1) in Listing 1.1; "distribute" means the transmission of meter readings via the network. For this reason, P1 is translated into a technical, machine-readable representation (P2) by using policy derivation mechanisms as described in [14][10]. Listing 1.3 depicts an excerpt of P2 for the Java abstraction layer. The policy specifies that each method invocation of *sendData()* (line 5) must be inhibited (line 10) if the method parameter *sendData.b* (line 6) contains meter readings from the file system directory */meter_readings* (line 7).

```
 1 <preventiveMechanism name="DoNotDistributeMeterReadings">
 2   <id>no_distribution</id>
 3   <trigger action="MethodInvoked" isTry="true">
 4     <paramMatch name="MethodName" value="sendData"/>
 5     <paramMatch name="ParamName" value="sendData.b"/>
 6     <paramMatch name="ParamValue" value="/meter_readings"/>
 7   </trigger>
 8   <condition> <true/> </condition>
 9   <action> <inhibit/> </action>
10 </preventiveMechanism>
```

Listing 1.3. Example policy: Do not distribute my meter readings.

To enforce policy P2, the PEP instruments every *iload*, *istore* and *invokestatic* bytecode instruction of Listing 1.2 with *PEP*-method invocations (cf. Listing 1.4) that reveal local variable assignments and method invocations within this code snippet. For instance, the instrumentation in lines {0,2,3} in Listing 1.4 invokes the method *PEP.beforeMethodInvoked()* (line 3) with two parameters that are pushed on the stack in lines {0,2} (these parameters being the string "readMr" and the integer "0"). These instructions reveal the information that the method *readMr()* is going to be executed. Method *readMr()* will then read a meter reading from the file system and push the result on the Java stack. Moreover, within

[10] To achieve all-pervasive usage control it is necessary to translate and enforce high-level usage requirements at all abstraction layers.

PEP.beforeMethodInvoked() the PEP requests a decision from the PDP (line 3) whether the execution of *readMr()* is allowed or *PEP.beforeMethodInvoked()* and returns `true` or `false` accordingly. Depending on this return value, the *ifeq*-instruction[11] (line 6) branches the program flow to line 15 (top stack value equals `false`) or executes the next instruction in line 9 (top stack value equals `true`). This way, *readMr()* is either bypassed or executed. In case of bypassing, the instruction in line 16 pushes the method's default return value onto the stack; the default return value depends on the method's return data type. In addition, the instrumentation in lines {18,20} signals that the instruction in line 23 stores the return value of *readMr()* in variable *meter_reading* (cf. line 2 in Listing 1.1).

```
0: ldc            #101   //String /readMr/p:1
2: iconst_0
3: invokestatic   #47    //Method PEP.beforeMethodInvoked:(Ljava/lang/String;I)Z
6: ifeq           15
9: invokestatic   #16  //Method readMr:()I
12: goto           17
15: nop
16: iconst_0
17: nop
18: ldc            #103 //String /main/meter_reading/1
20: invokestatic   #40  //Method PEP.storeVar:(Ljava/lang/String;)V
23: istore_1
24: ldc            #103 //String /main/meter_reading/1
26: invokestatic   #37  //Method PEP.loadVar:(Ljava/lang/String;)V
29: iload_1
30: ldc            #105 //String /main/b/2
32: invokestatic   #40  //Method PEP.storeVar:(Ljava/lang/String;)V
35: istore_2
36: ldc            #105 //String /main/b/2
38: invokestatic   #37  //Method PEP.loadVar:(Ljava/lang/String;)V
41: iload_2
42: ldc            #107 //String /sendData/b:2
44: iconst_0
45: invokestatic   #47  //Method PEP.beforeMethodInvoke:(Ljava/lang/String;I)Z
48: ifeq           57
51: invokestatic   #20  //Method sendData:(I)V
54: goto           59
57: nop
...
```

Listing 1.4. Instrumented bytecode of Listing 1.2

Instructions in lines {24,26} and {30,32} in Listing 1.4 reveal the information that data (here meter readings) are read from variable *meter_reading* (line 29) and are written into variable *b* (line 35); that corresponds to the variable assignment in line 3 in Listing 1.1. Finally, the instructions in {42,44,45,48} evaluate if the execution of method *sendData()* in line 51 is allowed or not; these instructions are analogous to the instrumentation in lines {0,2,3,6}. However, in this case the PDP's decision will evaluate to *inhibit*, as policy P2 forbids distribution of meter readings using method *sendData()*. More precisely, the method *PEP.beforeMethodInvoked* (line 45) receives an *inhibit* from the PDP and will therefore return `false`. The subsequent *ifeq*-instruction in line 48 succeeds and the program flow continues at line 57. This way, it is possible to enforce policy

[11] The *ifeq* bytecode pops the top stack frame value and compares it with zero. If the values are equal, the program flow continues at the specified offset; otherwise the bytecode instructions following *ifeq* are executed.

P2 and to track the (explicit) data flow in Listing 1.1 from method *readMr()*, via the variable assignment in line 3, and finally to method invocation *sendData()*.

In contrast to the PEP, the remaining components of our usage control infrastructure (PDP, PMP, and PIP) are independent of the abstraction layer and can be plugged to PEPs operating at arbitrary layers. For instance, to provide a usage controlled *Python* runtime environment, a python-specific PEP and data flow tracking mechanism must be implemented.

4 Security analysis

First of all, well-known cloud computing security issues apply to our cloud-based solution. These issues have been studied extensively before [26]. Hence, the following refers to the attack scenario in §1 and exposes some weaknesses and limitations of our usage control infrastructure and implementation.

Policy modification: An attacker could compromise the PMP and illicitly modify policies in order to reveal smart grid data, e.g. a modification from "do not distribute my meter readings" to "permit distribution of my meter readings". This way, desired information are released although the usage control infrastructure works correctly. To thwart such an attack, policies must be protected using digital signatures.

Integrity of usage control infrastructure: Our assumption of a trustworthy PaaS provider is not always reasonable. Hence, PaaS providers might compromise the integrity of the usage control infrastructure. For instance, a PaaS system administrator can exchange system or usage control components with compromised versions in order to circumvent usage control mechanisms. A possible countermeasure is the usage of *Trusted Computing (TC)* technology in order to generate cryptographic hash keys that ensure system integrity. Moreover, TC in combination with *remote attestation* facilitates the detection of changes on (remote) systems. This way, it is possible to send sensitive data only if the (remote) system can attest that it runs a valid usage control installation [13].

Data protection boundaries: A limitation of our data protection mechanism is that it is only capable of enforcing policies within the PaaS level. Once smart meter data leaves the controlled system boundaries, the infrastructure must ensure that either the receiving site (be it another system or another system layer) is also capable of enforcing usage control policies or that sensitive data are not released at all. In order to allow distributed policy enforcement, a distributed usage control infrastructure as described in [11, 12] must be incorporated.

Implementation concern: A non neglectable concern is the *Java Native Interface (JNI)*. An attacker might circumvent our usage control infrastructure by providing his own classes that perform JNI calls in order to realize similar or the same functionality as the Java system-classes. For instance, if a usage control policy inhibits the invocation of the system-class method *DatagramSocket.send()*, an attacker might provide an own class, e.g. *MyDatagramSocket*, that performs JNI calls in order to enable UDP communication. A possible countermeasure is to make use of cross-layer data flow tracking mechanisms [22] in order to track the data flow via Java JNI calls to underlying abstraction layers, e.g. the operating system layer. Usage requirements can then be enforced on these underlying layers.

5 Related Work

Smart grid vulnerabilities as well as security and privacy issues have been exposed extensively both from legal [6, 8] and technical [4, 5] perspectives. Yet, concerns regarding the abuse of smart grid metering data persist, since meter readings can provide inferences of users' behavior.

[7] suggests to send low-frequency metering data (collected over a longer period, e.g. over several weeks) to the utility company directly, whereas high-frequency metering data must be anonymized beforehand by an escrow service.

[10] assumes that future smart homes will contain both energy generation and storage devices (e.g. batteries). On this basis the authors propose an algorithm for maintaining a constant meter load by discharging or recharging the battery, hence hiding information about users' behaviour.

The authors of [16] present an approach of extracting usage patterns from smart meter data. Also, they propose a privacy-enhanced smart meter architecture and describe a zero-knowledge protocol for billing issues.

[24] presents a cloud computing model for managing smart grid real-time data streams. The authors propose to protect privacy by data aggregation.

[1] presents a privacy-preserving model based on aggregation and encryption mechanisms. The proposed model adds noise and encrypts smart meter readings before they are released, thus providing differential privacy with low error. However, such errors are not acceptable for billing purposes.

In contrast to these works, our approach is not primarily based on data obfuscation and data encryption. Instead, we aim at controlling the usage of data within defined system boundaries by enforcing user defined data usage policies. Our infrastructure ensures that malicious software running within these system boundaries complies with these policies.

6 Conclusion and future work

With the advent of the smart grid a plenty of energy prosumers' privacy concerns arise because research has shown that it is possible to deduce private details of residents's lifestyle from (fine-grained) meter readings [16], especially when those are released to (cloud-based) thrid-party smart grid services. To mitigate this concern, we have introduced a new data protection mechanism for cloud-enabled smart grid services by deploying usage control concepts and mechanisms for the cloud PaaS model. We have described the usage control components, their functionalities, and their integration with smart grid services. Additionally, our approach is motivated by taking an energy marketplace as a smart grid service example. Our usage control mechanisms allow users to control the usage of their sensitive data, particularly smart meter readings, once they have been given to smart grid services. As proof of concept we show how these mechanisms can be provided at the cloud PaaS level using a Java runtime environment.

Our integration of usage control is based on the assumption of a trustworthy PaaS provider. As this assumption is not always reasonable because a malicious PaaS provider might circumvent our usage control mechanism, it is necessary to introduce usage control for all data abstraction layers within the PaaS provider. At the moment, usage control implementations exists for different data abstraction layers, e.g. for the operating system (Windows, OpenBSD, Android, ChromeOS) or the browser (Firefox) layer.

In terms of future work, it is not entirely clear how policy derivation mechanisms must deal with the heterogeneity and dynamics of software systems when translating high-level usage requirements into low-level technical requirements (cf. [14]). Furthermore, it is not clear how to solve the problem of overapproximation. For instance, once smart meter data are involved into a price-calculation, the result of this calculation will be usage controlled as well, because it is not obvious if and how much information of smart meter data are included in the output. In a worst case this approach leads to an unusable system after some time. We are currently working on appropriate declassification mechanisms. Finally, further open questions are: how guarantees can be measured, how the usage control infrastructure can be secured, and how the existence of usage control mechanisms can be ensured.

References

1. Ács, G., Castelluccia, C.: I Have a DREAM! (DiffeRentially privatE smArt Metering). In: Information Hiding, vol. 6958. Springer Berlin / Heidelberg (2011)
2. BMWi, T.C.: PeerEnergyCloud, http://www.peerenergycloud.de/
3. BMWi, E-Energy: eTelligence Project, http://www.etelligence.de
4. Cavoukian, A., Polonetsky, J., Wolf, C.: SmartPrivacy for the Smart Grid: embedding privacy into the design of electricity conservation. Identity in the Information Society 3, 275–294 (2010)
5. Clements, S., Kirkham, H.: Cyber-security considerations for the smart grid. In: Power and Energy Society General Meeting, 2010 IEEE. pp. 1–5 (Jul 2010)
6. Eckert, C., Krauß, C.: Sicherheit im Smart Grid - Herausforderungen und Handlungsempfehlungen. Datenschutz und Datensicherheit - DuD 35, 535–541 (2011)
7. Efthymiou, C., Kalogridis, G.: Smart Grid Privacy via Anonymization of Smart Metering Data. In: 1st IEEE Intl. Conf. on Smart Grid Communications (Oct 2010)
8. Farhangi, H.: The path of the smart grid. Power and Energy Magazine, IEEE 8(1), 18–28 (Jan 2010)
9. Gong, C., Liu, J., Zhang, Q., Chen, H., Gong, Z.: The Characteristics of Cloud Computing. In: 39th Intl. Conf. on Parallel Processing Workshops (Sep 2010)
10. Kalogridis, G., Efthymiou, C., Denic, S., Lewis, T., Cepeda, R.: Privacy for Smart Meters: Towards Undetectable Appliance Load Signatures. In: First IEEE International Conference on Smart Grid Communications. pp. 232–237 (Oct 2010)
11. Kelbert, F., Pretschner, A.: Towards a Policy Enforcement Infrastructure for Distributed Usage Control. In: Proc. 17th ACM Symp. on Access Control Models and Technologies (Jun 2012)

12. Kelbert, F., Pretschner, A.: Data Usage Control Enforcement in Distributed Systems. In: Proc. 3rd ACM Conference on Data and Application Security and Privacy. CODASPY '13, ACM (Feb 2013), to appear.
13. Kumari, P., Kelbert, F., Pretschner, A.: Data Protection in Heterogeneous Distributed Systems: A Smart Meter Example. In: Proc. Workshop on Dependable Software for Critical Infrastructures. GI Lecture Notes in Informatics (Oct 2011)
14. Kumari, P., Pretschner, A.: Deriving implementation-level policies for usage control enforcement. In: Proc. 2nd ACM Conf. on Data and Application Security and Privacy (2012)
15. Mell, P., Grance, T.: The NIST Definition of Cloud Computing. Tech. rep., National Institute of Standards and Technology (Sep 2011)
16. Molina-Markham, A., Shenoy, P., Fu, K., Cecchet, E., Irwin, D.: Private memoirs of a smart meter. In: Proc. 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building. pp. 61–66 (2010)
17. National Energy Technology Laboratory for the U.S. Department of Energy: Advanced Metering Infrastructure. Tech. rep., U.S. Department of Energy (Feb 2008)
18. Ockwell, G.: The DOE's "7 Traits of a Smart Grid". Fortnightly's Spark (Oct 2009)
19. Park, J., Sandhu, R.: The UCONABC usage control model. ACM Trans. Inf. Syst. Secur. 7(1), 128–174 (Feb 2004), http://doi.acm.org/10.1145/984334.984339
20. Pretschner, A.: An Overview of Distributed Usage Control. In: Proc. 2nd Conf. Knowledge Engineering: Principles and Techniques. Romania (Jul 2009)
21. Pretschner, A., Hilty, M., Basin, D., Schaefer, C., Walter, T.: Mechanisms for Usage Control. In: Proc. 2008 ACM Symposium on Information, Computer and Communications Security. pp. 240–244 (Mar 2008)
22. Pretschner, A., Lovat, E., Büchler, M.: Representation-Independent Data Usage Control. In: Data Privacy Management and Autonomous Spontaneus Security, vol. 7122, pp. 122–140. Springer Berlin Heidelberg (2012)
23. Quinn, E.L.: Smart Metering & Privacy: Existing Law and Competting Policies. Tech. rep., Colorado Public Utilities Commission (2009)
24. Rusitschka, S., Eger, K., Gerdes, C.: Smart Grid Data Cloud: A Model for Utilizing Cloud Computing in the Smart Grid Domain. In: First IEEE International Conference on Smart Grid Communications. pp. 483–488 (Oct 2010)
25. Simmhan, Y., Aman, S., Cao, B., Giakkoupis, M., Kumbhare, A., Zhou, Q., Paul, D., Fern, C., Sharma, A., Prasanna, V.: An Informatics Approach to Demand Response Optimization in Smart Grids. Tech. rep. (2011)
26. Subashini, S., Kavitha, V.: A survey on security issues in service delivery models of cloud computing. Journal of Network and Computer Applications 34(1) (2011)
27. Wetzel, Daniel: Ende der Subventionen (Sep 2011), http://www.welt.de/print/die_welt/wirtschaft/article13603359/Ende-der-Subventionen.html