CLOUD ACCOUNTABILITY PROJECT

D:D-3.1: Enforcement Tools, Service Specification and Architectural Design

Deliverable Number: D43.1
Work Package: WP 43
Version: Final
Deliverable Lead Organisation: SAP
Dissemination level: PU
Contractual Date of Delivery (release): 30/11/2013
Date of Delivery: 20/12/2013

Editors
Jakub Sendor (SAP)

Contributors
Anderson Santana de Oliveira (SAP), Alexander Garaga (SAP), Kateline Jenatton (SAP), Jakub Sendor (SAP), Monir Azraoui (EURC), Kaoutar Elkhiyaoui (EURC), Melek Önen (EURECOM), Walid Benghabrit (EMN), Jean-Claude Royer (EMN), Mohamed Sellami (EMN), Nick Papanikolaou (HP Labs)

Reviewers
Leonardo Martucci (KAU), Tomasz Wiktor Wlodarczyk (UiS)

A4Cloud
www.a4cloud.eu
Accountability For Cloud and Other Future Internet Services

FP7-ICT-2011-8-317550-A4CLOUD

SEVENTH FRAMEWORK PROGRAMME

Contents

List of Figures 4

Executive Summary **5**

1 Introduction to Policy Enforcement Framework **7**
1.1 A General View of Policy Enforcement 7
1.2 Review of the Existing Enforcement Methodologies 9
1.3 Obligations Mapping 10

2 Extensions to the Existing Tools and Services **13**
2.1 PPL Engine 13
2.1.1 Core Engine Components 13
2.1.2 Event and Obligation Handlers 15
2.2 Interfaces Required for Integration 16
2.2.1 Secure Logging 16
2.2.2 Evidence Collection 17
2.2.3 Audit 18
2.3 Extended Policy Enforcement Engine Architecture 18

3 Data Transfer Control Toolkit: Architecture and Design for The New Enforcement Tools **21**
3.1 Motivation for Data Transfer Control in the Cloud 21
3.2 Introduction to Data Transfer Control Toolkit 22
3.3 Data Tracking Challenges in the Cloud 23
3.4 Data Tracking Monitors 23
3.5 SaaS and PaaS Monitoring 24
3.6 IaaS Monitoring 26
3.7 Topology 28
3.8 Accountability Service 29
3.9 Audit Trails 30
3.10 Security Considerations 31

Conclusions **33**

A Enforcement Methodologies Survey **35**

References **38**

List of figures

1.1 General Policy Framework introduced by WP C-4 8
1.2 Design Time for Accountability Policies 8
1.3 Run Time Enforcement 9
1.4 AccLab Architecture 10

2.1 PPL Engine Architecture 14
2.2 Logging with Non-Repudiation 17
2.3 Audit 19
2.4 PPL Engine with the interfaces for the extension 19

3.1 Data tracking architecture 22
3.2 Data Tracking Monitor Architecture 24
3.3 Accountability service architecture 29

Executive Summary

Automatic policy enforcement requires provisioning of a toolkit allowing to model the necessary policy statements and to guide the processes of ensuring compliance with the policies. The A4Cloud policy enforcement and compliance toolkit needs to built on top of a consistent policy specification and enforcement methodologies, considering the already existing enforcement solutions in the cloud landscape, such that we can maximize its adaptability to cloud environments, and thus facilitate its integration with cloud services.

The A4Cloud policy framework, introduced by WP C-4 ("Policy mapping and representation") [GdOS+13] defined a policy model where regulatory, contractual, security and privacy concerns can be expressed, with particular emphasis on constraints about personal and business confidential usage. Accountability policies are enforceable across the cloud service provision chain by means of accountability services and tools.

In this document, we will present the design of the accountability toolkit establishing the feasibility of the tools supporting the methodologies identified in the policy framework that is delivered by WP C-4:

- Chapter 1 introduces the A4Cloud policy framework that was defined in the WP C-4 and describes AccLab tool (which stands for Accountability Laboratory) that provides support for writing abstract accountability obligations, checks their consistency and compliance, and ultimately transforms them into A-PPL policies for enforcing obligations via the A-PPL engine.

- Chapter 2 deals with the existing tools that support the enforcement mechanisms envisioned in the policy framework. We present the PPL Engine, which is the policy enforcement engine that was designed in the scope of the PrimeLife project [Pri11]. We identified the extension points which will make it possible to plug-in the necessary functionality introduced by the policy framework: robust logging, auditing and evidence collection.

- Chapter 3 contains the design of the data transfer control tools for the cloud, with the architecture blueprint and a working example in a selected A4Cloud use case [BOS+13]. The approach automates the collection of evidence that obligations concerning personal data transfers are being carried out properly. The rules determining the obligations can be obtained from the policies (enforceable by the A-PPL Engine) and thanks to the configuration provided by the user are evaluated together with the data transfer logs to check for any policy violation.

- Additionally we provide in Appendix A an early study conducting a survey of the general enforcement methodologies that abstracts from the A4Cloud policy framework. This was a preliminary work that established the policy enforcement principles (in terms of tooling) for this work package, before the WP C-4 will supply the report on the policy enforcement techniques (which is planned to be delivered in Month 24).

As this report aims mainly at building the ground for the practical implementation tasks scheduled to start in Month 16, the nature of this document is in most of the parts a technical documentation of the tools that will constitute the policy enforcement framework. We advise the readers interested in the detailed description of the A4Cloud policy framework to refer to the deliverable on the policy representation framework [GdOS+13].

Chapter 1

Introduction to Policy Enforcement Framework

In this chapter we first present in Section 1.1 a general view of the policy enforcement tools, based on the A4Cloud policy framework introduced in [GdOS+13]. Later on in Section 1.2 we discuss this initial view, drawing also the conclusions from our study summarized in Appendix A. Section 1.3 closes this chapter with the architecture of the AccLab, the tool that will be enforcing AAL (Abstract Accountability Language) obligations introduced by WP C-4.

1.1 A General View of Policy Enforcement

Accountability policies will be defined at first in a higher abstraction level, and then translated into enforceable policies as depicted in Figure 1.1 [GdOS+13]. In order to accomplish the translation from the abstract accountability policy language into concrete policies, we introduce the AccLab tool. This tool helps in writing abstract accountability obligations, to check for consistency and compliance and then it generates A-PPL policies for enforcing the obligations via the A-PPL engine. It enables the user to experiment with accountability obligations and to evaluate their adequacy. The input is provided via a smart wizard which helps in writing AAL obligations (for a privacy officer) and user preferences (for a data subject).

The tool internally represents the obligations and computes A-PPL policies for the enforcement of these obligations. It is also devoted to check consistency that is to find some errors in the obligation description. The compliance check allows to establish if one obligation is stronger than (takes the precedence over) another obligation.

The tool will have three modules, the first for visually declaring system resources (agents, services, data, etc.), the second for the wizard with lists of choices to write obligations, the third for an AAL code editor, and a toolkit with different action set, e.g. to check consistency, compliance and to generate A-PPL (Figure 1.2).

The output will be consumed by the A-PPL engine, thus it should conform to the A-PPL schema (see Figure 1.3). The main purpose of AccLab is however to facilitate

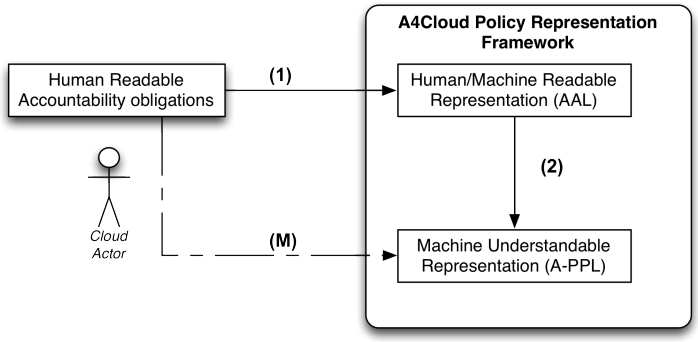

Figure 1.1: General Policy Framework introduced by WP C-4

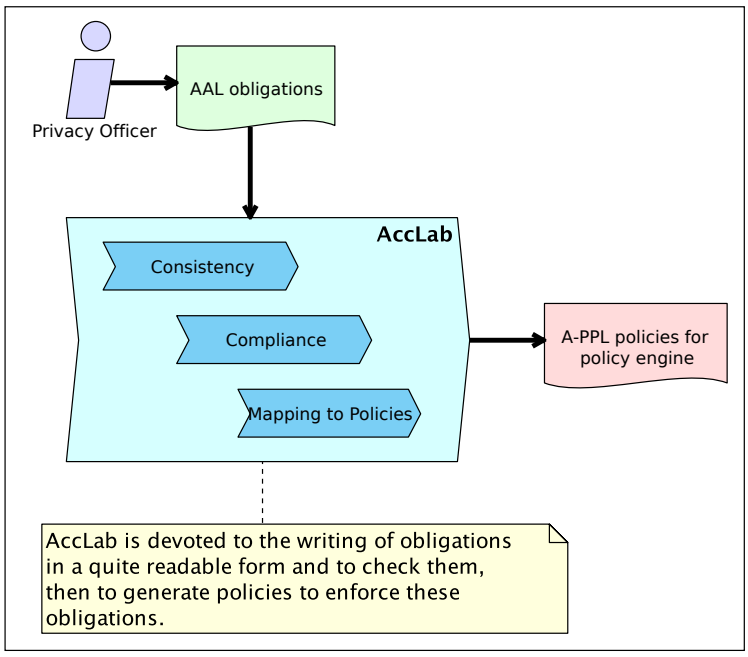

Figure 1.2: Design Time for Accountability Policies

the obligations writing process in a form that is readable, to check them, and to generate policies to enforce these obligations. In that sense, the tool can be seen as an intermediary step in policy enforcement, taking accountability obligations as its input and producing enforceable A-PPL policies that are stored in a Policy Repository, to be used by the A-PPL Engine during the actual policy evaluation process.

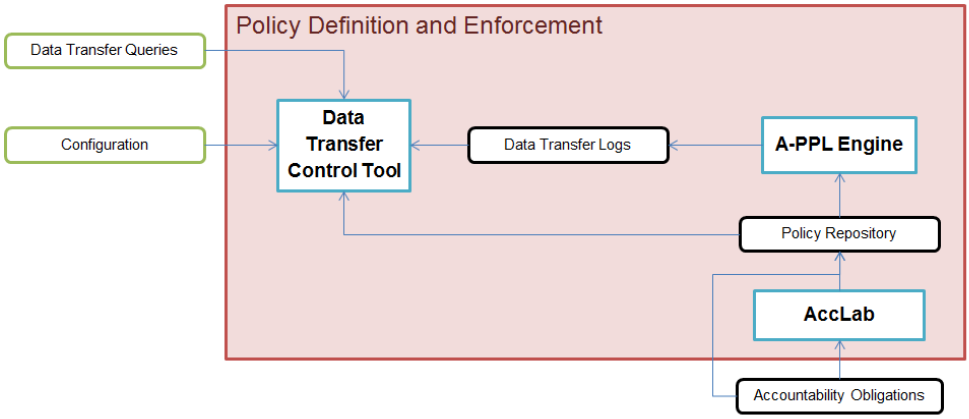

Figure 1.3: Run Time Enforcement

1.2 Review of the Existing Enforcement Methodologies

From our survey of the existing tools summarized in Appendix A we can draw some conclusions. There are few existing frameworks covering a holistic view of accountability. Often these are academic prototypes demonstrating the feasibility of some solutions. The descriptions are generally limited to few academic papers and no existing tool support can be reused. There are some possible tools for secure logging and auditing for instance, but still we need to interface with them. In our survey there are several approaches related to the methodology for enforcement. Of course we can reuse some concepts, ideas and architectures from these approaches, but regarding precise tools it is less obvious. [HKK12] assumes a trusted cloud infrastructure for policy enforcement and consider machine readable policies mainly safety policies enforced by inline reference monitors. It does not take into account the full attributes of an accountability policy. With [BKF+11] we have a more abstract view of policy with an automatic refinement matching. But still the concept of accountability is lacking and it is not obvious how to extend the proposal to solve it. It relies on the Resource Description Framework and associated tools which should be extended to cover access, usage control and accountability. In our review we have not found a general and effective methodology to enforce accountability policies. As shown in the state of the art in the D34.1 deliverable [GdOS+13], concrete policy languages are rather seldom. We expect to extend PPL since it already provides a good set of concepts like usage control, notification, log, etc. In fact most of the existing frameworks assume predefined user preferences or system obligations and expect to enforce them. But here we need a more generic approach allowing the Data Subjects, Data Controllers, auditors or others to define their preferences and obligations. Thus we need an architecture and a methodology taking into account this genericity. We will see later in Section 1.3 our proposed architecture for the enforcement of accountability obligations.

1.3 Obligations Mapping

In this section we give our methodology and architecture to achieve enforcement of accountability obligations.

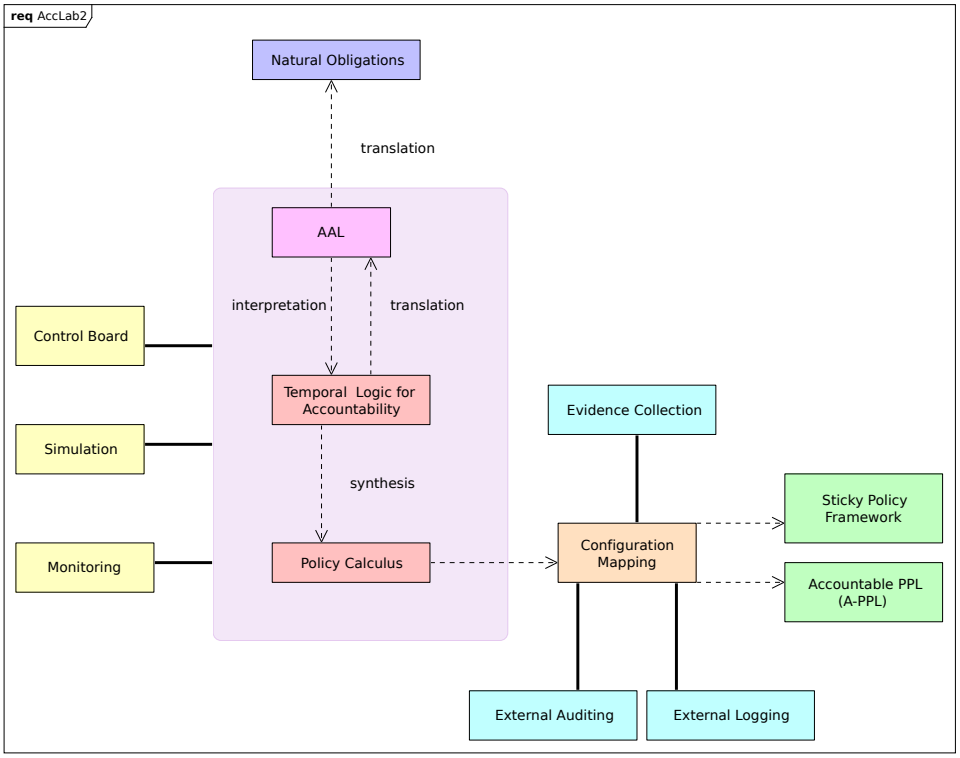

Figure 1.4: AccLab Architecture

Figure 1.4 presents our mapping architecture elaborated from the analysis and the proposal made in [GdOS+13]. The kernel part of the AccLab consists of: AAL, Temporal Logic for Accountability and Policy Calculus modules. The input is the AAL module which represents an entry point for a precise language to write accountability obligations. The concrete targets for policy enforcement are represented by the components at the right bottom of the diagram (Sticky Policy Framework and A-PPL).

We first describe the usages we envision:

- Researchers will tune the various languages and relations using the AccLab prototype. It can be used to check the readability of the AAL and also the suitability of the current constructions. It will be used to experiment with some required extensions, for example time expressions.
- Data Subjects but also Data Controllers or Data Processors can use it to write their user preferences and obligations. It will be possible to check the consistency

of these obligations as well as the compliance of a user preference with a provider obligation.

- Data Controllers can use the simulation or monitoring facilities to predict or observe the behavior of their system with accountability requirements. This means that when the system is supplied with obligations and a simulation of the system is launched, the observer can see whether the system is correctly executing the accountability requirements. Firstly, a description of the system to simulate should be provided in terms of agents interacting with messages. This consists in a structural description of the agents and their communications, and also scripts to represent the agent behaviors. Secondly, accountability obligations, written in AAL, should complete the system model. This facility could be used by the Data Controller to tune his AAL descriptions, to track bugs or misbehavior and even to validate an accountable design.

- Data Controllers and Data Processors can use it to generate concrete policies in a PPL style or in another concrete accountability framework. The kernel is devoted to be a pivotal system enabling translation and configuration to several targets.

The AAL, Temporal Logic for Accountability, Policy Calculus and A-PPL components represent processing modules for the different concrete languages which are described with much more details in the D34.1 deliverable [GdOS+13]. Here we will focus on the tools perspective and the enforcement of policies.

- AAL: the AAL represents the main input as a machine readable language for obligations.
  - The translation into the obligations specified in natural language describes a simple rephrasing algorithm to help non specialist in reading AAL expressions. This is clearly not sufficient for a good readability, other ideas are discussed in [GdOS+13] but this is not the focus of the current deliverable.
  - The interpretation to Temporal Logic for Accountability is a translation into a formal language with a temporal logic and notions of resources and agents. It formally defines the property denoted by the AAL expression.

- Temporal Logic for Accountability: from this module there are two connections. The translation is the reverse process of the interpretation, this is a simple way to provide something more readable after simplification or rewriting of logic expressions. The synthesis consists in generating a program, in the Policy Calculus, starting from a temporal property. This temporal property should be ensured by the program, that is any execution of the program should satisfy the logical property. Synthesis is a classic concept in formal verification[1] and often associated with logical properties as we have here.

[1] http://en.wikipedia.org/wiki/Program_synthesis

- Policy Calculus: it is an abstract but operational calculus including security expression, accountability, reference monitors and resources. It is expected to be executable for simulation but also to be mapped to the more concrete languages, like A-PPL, by Configuration Mapping. The Policy Calculus should be viewed as a pivotal language to express and enforce accountability concepts.
- Configuration Mapping: this module represents mainly the process of mapping the Policy Calculus to a more concrete policy language for effective enforcement. Such mapping translates the Policy Calculus into the concrete language constructions dealing with the policy enforcement. Thus the process should be defined for each concrete policy language we expect to target, in our case it is A-PPL. Since we expect to reuse existing tools, this module allows the configuration of the mapping to take into account: secure logging, evidence collection and auditing. But it could also cope with other parameters related to specific enforcement framework characteristics.
- Accountable PPL (A-PPL): this is our primary target for the policy enforcement. The exact way to translate AAL sentences into A-PPL is not yet defined. There are a set of sentences which should be translated in a direct way: access control expressions, some data transfers and certain obligations. However, it is not so obvious for AAL clauses involving temporal constraints and complex data disclosures. The main idea is to do a projection of the AAL clauses to each agent and resource in order to obtain expressions an endpoint can enforce.
- Sticky Policy Framework: it represents an alternative to PPL that will be designed and implemented later in the project. We are planning to investigate cryptographic schemes for implementing sticky policies for accountability. As part of the WP D-3 we are planning to implement a demo of the stickiness aspect, namely the cryptographic scheme that prevents data from being accessed unless the policy attached to it is satisfied. The sticky policy framework is independent of the actual policy language and operates at the lowest level of enforcement, so this work can proceed without affecting the PPL Engine extensions.
- Control Board: it represents a tool set to define agents with their services and to configure the kernel.
- Simulation: this consists in executing a Policy Calculus program and to observe accountability in action. The exact presentation of the result and the interactions with the user are not yet defined.
- Monitoring: while it is not our primary task it could provide a way to monitor real system. For each real agent we want to monitor we create a fake agent in the laboratory whose role is to control the real agent. The fake agents in the kernel delegate their actions to the real agent of the running system. Since our calculus relies on reference monitor this control is direct to implement. However, one problem is to get the control on real agents, this raises a difficult question but it is out of the scope of this deliverable.

Chapter 2

Extensions to the Existing Tools and Services

In this chapter we will provide overview of the existing support mechanisms for PPL, which was identified by the deliverable D34.1 [GdOS+13] as a candidate policy language for the extension with the accountability features. The language is based on the access control standard XACML with additional elements related to the usage control.

2.1 PPL Engine

Privacy policy engine supporting PPL was designed in PrimeLife project [TNR11]. The engine supports the scenario defined as an interaction between Data Subject (DS) and Data Controller (DC), in which Data Subject is sharing his personal data with Data Controller in order to gain access to the certain services provided by the latter. After initial DS request DC returns the service policy. This policy is matched with DS preferences and in case of positive match DS personal data is released to the DC together with a privacy policy. Such policy, a result of matching between DC service policy and DS preferences, is attached to the piece of personal data and stored together in the Personally Identifiable Information (PII) repository along with this data. The policy is referred to as a Sticky Policy in the PPL Engine specification.

The components in the Figure 2.1 depict the policy engine architecture that corresponds to the described scenario.

2.1.1 Core Engine Components

The core elements of the policy engine are the components in the Business Layer: Policy Enforcement Point (PEP) and Policy Decision Point (PDP). While the PEP acts as an orchestrator of the enforcement process and interface with the Web Server (which together with the elements in the Presentation layer is PrimeLife scenario specific and we will omit it in further description) the PDP is the component where the access control decision is taken, as well as policies and preferences are being matched. The matching

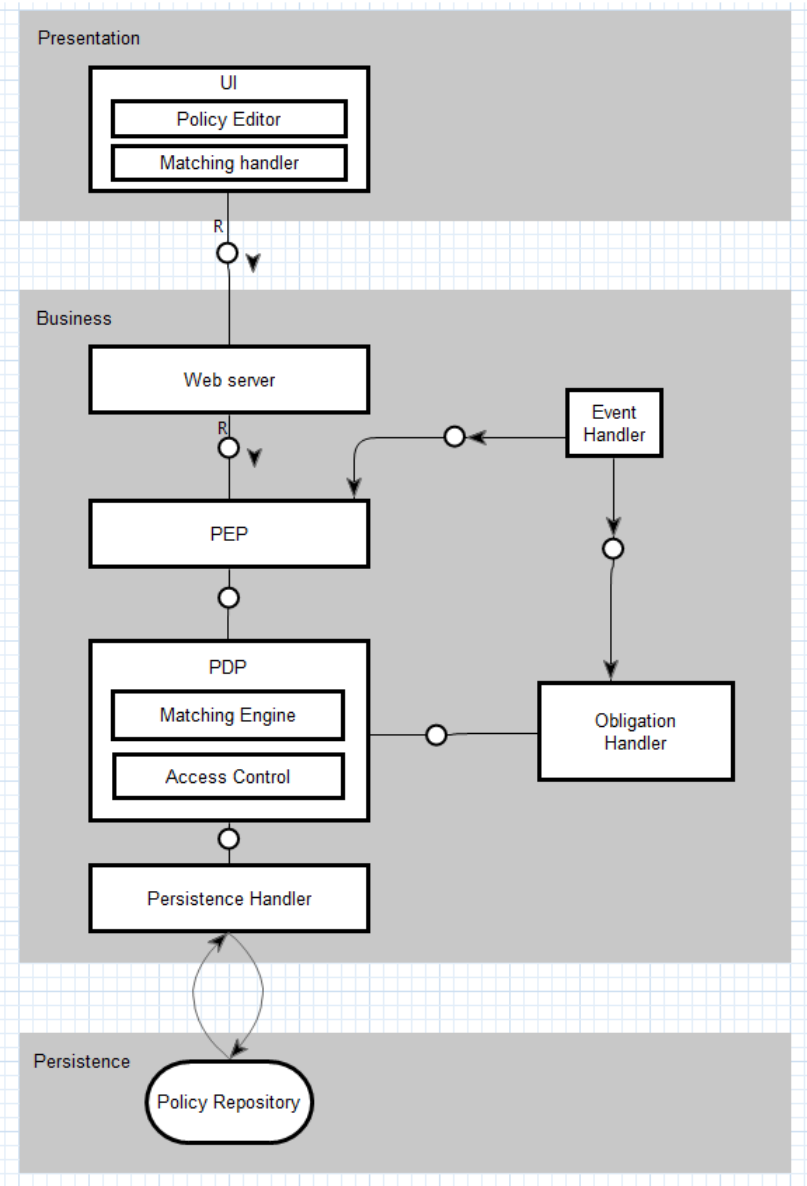

Figure 2.1: PPL Engine Architecture

procedure, interesting from the PrimeLife project objectives point of view, was the main concern in the implementation of the engine, however it is less significant in the cloud scenario that is in focus of A4Cloud.

PDP relies on the access control engine implementation based on HERAS [HER] for the evaluation of XACML part of PPL policy. Apart from the standard attribute-based access control, the other information evaluated by the PDP at the step of access control decision is usage authorization and the result of policy matching. The usage authorization basically consists of the comparison of the list of purposes specified in

the Data Subject preferences with the one specified by Data Controller in his policy. It also compares the authorization for the downstream usage (use of the data by the third parties, with whom Data Controller might share the collected data in the future).

As the personal data is stored together with the associated privacy policy in the Database, the PDP is communicating with the PII Store/Policy-Preference Store by the Persistence Handler interface.

2.1.2 Event and Obligation Handlers

The PEP orchestrates two modules: the Event and Obligation Handlers. The functionality of the Event Handler is to fire the events related to the personal data lifecycle, e.g. when data is deleted from the PII store or when it is shared with the third parties.

The Obligation Handler is the component responsible for the execution of the obligations that form the part of the usage control restrictions defined in the Sticky Policy. Therefore it is initialized after the successful policy matching. It is the responsibility of the Obligation Handler to keep track of the triggers that are part of the obligation statements in the PPL.

Once the events are observed, which might be the case of receiving the notification from the Event Handler for the event-based triggers or simple time-outs in the case of the time-based triggers, the action associated with the obligation is activated by the Obligation Engine.

The Obligation Engine specification does not propose how certain types of actions are to be implemented. For instance, the obligations related to the secure logging are not precisely defined and are left open for the implementation in the specific scenario. In the next section we will try to address the necessary extensions for the PPL Engine architecture coming from the new language elements proposed in A-PPL [GdOS$^+$13].

2.2 Interfaces Required for Integration

So far we have highlighted the main elements of the privacy policy engine implementing the PPL specification. The extension to the PPL proposed by the WP C-4 (referred to as A-PPL) defines richer syntax and handling mechanisms in order to provide the accountability property for the cloud systems dealing with personal data. In this section we will take a closer look at the interfaces which can be used in PPL Engine as the extension points for the new functionality.

We will analyze each of the extension proposed as well as the effort required to implement it in the extended engine architecture. The interfaces necessary to integrate the new components are sketched in the following sub-sections. We will also provide an updated architecture diagram containing how the policy engine integrates with the new components.

2.2.1 Secure Logging

In order to provide extensible secure logging interface for the PPL engine, we propose to introduce a new component into the overall architecture which will take care of all logging related functionality during the policy enforcement and personal data handling process. This new component, which we will refer to as Logging Handler, provides interfaces that other components, mainly Obligation Handler, can use to register that a certain action related to the personal data lifecycle took place. In addition to the classical functionalities of secure logging, we will augment this Logging Handler with an *optional security feature* that ensures that a Cloud Consumer cannot deny that it requested the execution of some operations on its behalf and that the Cloud Provider cannot repudiate that it agreed to execute these operations. It is important to note that this additional functionality assumes that the Cloud Provider and the Cloud Consumer are never trusted, and it can easily be employed in the case of dispute between the Cloud Provider and the Cloud Consumer. We will refer further to this security feature as “Logging with non-repudiation”.

Now to implement this secure logging interface, we will define in addition to the Logging Handler, a Crypto Module that will perform the cryptographic operations that one will need to ensure basic secure logging operations together with the non-repudiation functionality.

Logging With Non-Repudiation

To achieve logging while assuring non repudiation, we propose to implement a fair exchange protocol that involves both the Cloud Provider and the Cloud Consumer and relies on message exchanges between these two entities. At the first phase of this exchange, the Cloud Consumer requests the logging of one or several consecutive events by sending an undeniable proof of that request. The Cloud Provider should first verify this proof: therefore the Logging Handler contacts the Crypto Module to verify the signature of the initial request. If the signature is correct then the Logging Handler logs

the required action or set of actions and sends an undeniable signature to the Cloud Consumer with the help of the Crypto Module again.

As also defined in [Ate], to achieve fairness, that is, to allow the Cloud Consumer to receive the proof of logging actions and prevent him to deny his request, the previously sent messages were in fact encrypted. Indeed, signatures alone do not guarantee fairness since the Cloud Provider could deny having received a certain signature. Therefore, entities send their signatures encrypted (but verifiable) and during the second phase of the protocol, each entity sends the decrypted version of the signatures which is considered as the original proof of requests and logs. Thanks to this second phase, the Cloud Provider can show that it has received the Consumer's consent for the logging of that specific event or set of events, while the Cloud Consumer has an undeniable proof that the Cloud Provider agreed to log those events. In case of disputes, any entity can contact a trusted third party (for example, the Auditor) which can decrypt any of the previously sent messages and identify the malicious entity.

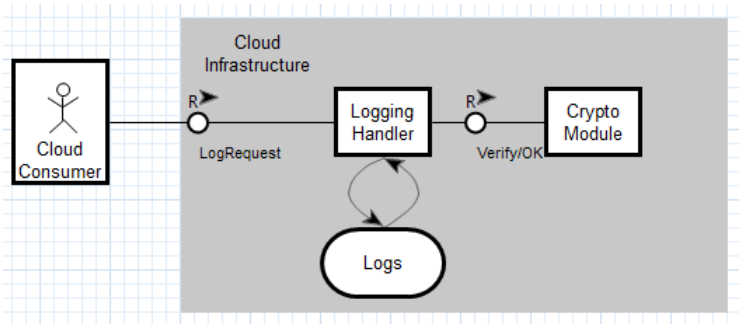

Figure 2.2: Logging with Non-Repudiation

Figure 2.2 illustrates the new components and communication channels that are used in the described scenario.

2.2.2 Evidence Collection

Some accountability scenarios require the collection of the evidences defined by the policy itself. Work Package C-8 is currently working on the definition of different evidence types. For example, Cloud Consumer may want to ensure transparency in service delivery chains and asks the Cloud Provider to provide data transmission timestamps for communication among providers.

On the other hand, a Cloud Consumer may wish to request some evidence on the retrievability of the data. In order to tackle these scenarios in the policy enforcement framework, we propose to define a new Evidence Collector component, which based on the policy, collects the evidences required for any accountability action.

As the work regarding the evidence collection is at the early stage, we defer from providing an architecture of this component here. We propose that the Evidence Collector interacts with the Obligation Handler component from the PPL Engine.

2.2.3 Audit

Language extension in A-PPL related to audit is one of the requirements that has to be taken into account by the proposed engine architecture. We plan to cope with it by extending the initial PPL engine architecture with the components facilitating the auditing procedures.

The core functionality allowing auditing of the actions taken by the engine is the Logging Handler component described earlier in this chapter. Additional considerations has to be also taken with respect to the Obligation Handler module, that manages the enforcement of the policy statements related to the Data Controller obligations, as well as the Event Handler component, which executes the data related obligations.

Also the central modules responsible for the evaluation of the policy and the policy enforcement (PDP and PEP) need to provide a means to observe, control and check, whether the execution of the process is correct and corresponds to the policy and the policy language specification, which highlights how the decision process should take place.

Therefore, we propose to add a central component for handling the audit requests, which will facilitate the process of retrieving the necessary information from the systems (logs related to obligations, notifications, access control decisions and personal data lifecycle). Furthermore, each component in PPL architecture that is related to this information (Obligations Handler, Event Handler, PDP and PEP) will be extended with the logging adapter that will make it possible to record all data sensitive actions in a non-repudiable manner. The adapter will need to implement simple interface, based on the interaction with the Logging Handler from Section 2.2.1.

By accessing the central Audit component, the auditors (internal or external), can gain the insight into the operations of the engine and correlate these information with the policies that were deployed. The component also needs to make sure that information retrieved from the audit logs does not leak the personal data collected in the cloud system nor any other business sensitive information. Yet it should allow the auditors to simplify the often cumbersome and time-consuming task of collecting all required input from a software system. It might be also envisioned that evidences collected by the component introduced in Section 2.2.2 will also contribute to the auditing process.

We have depicted the architecture elements that will take part during the external audit in Figure 2.3. To simplify the orchestration of this process, we will allow only three components to provide the input: Obligation Handler, Logging Handler and Evidence Collector. The information provided by all other components (PDP, PEP and Event Handler) is going to be already accessible via Logging Handler.

2.3 Extended Policy Enforcement Engine Architecture

The elements proposed in this section are a building blocks for the new enforcement functionality that was proposed in the accountable version of PPL. They can be easily integrated into the PPL Engine thanks to the extensibility of the generic components like Obligation Handler.

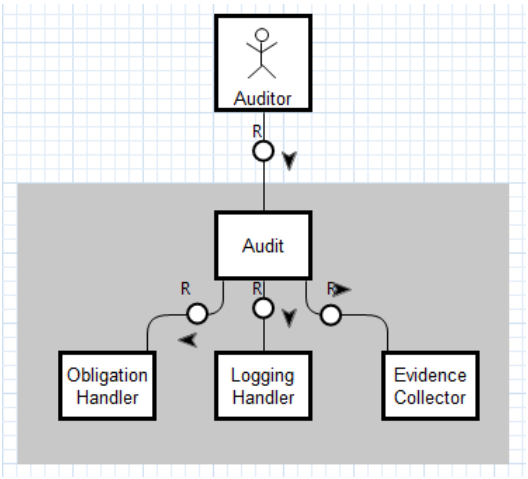

Figure 2.3: Audit

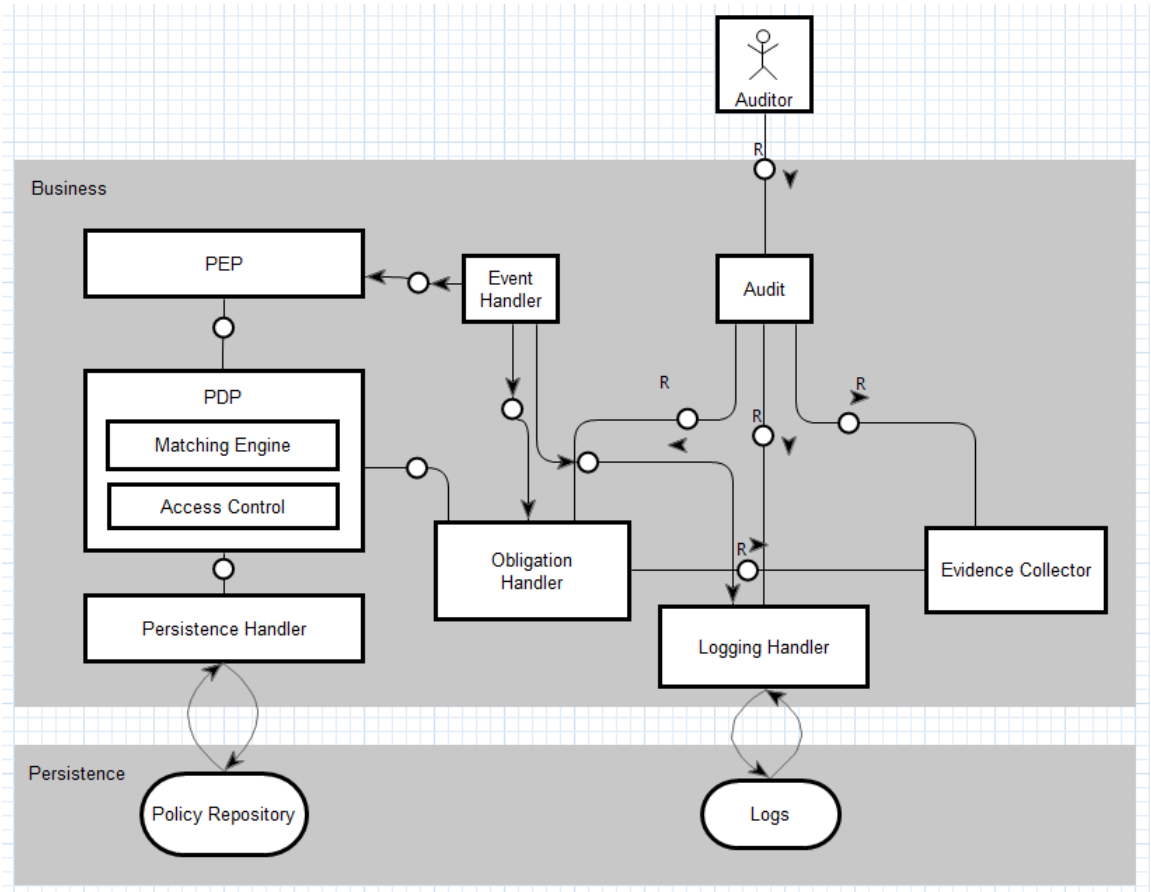

Figure 2.4: PPL Engine with the interfaces for the extension

In Figure 2.4 we show how the new components fit into the architecture described in Section 2.1. Apart from the implementation of the Audit component, which will require necessary changes in all of the initial engine modules, we can assume that the rest of the extensions would do not impose so severe modifications. They will most likely resemble a standalone, pluggable elements of a larger accountability toolset.

Chapter 3

Data Transfer Control Toolkit: Architecture and Design for The New Enforcement Tools

We will focus now on the design of the additional tools we plan to provide in this work package, especially related to the controlling data transfer in the cloud, that is a second goal of this work package. Together with the AccLab from Chapter 1 and the PPL Engine extensions described in Chapter 2 they will complete the robust policy enforcement framework.

3.1 Motivation for Data Transfer Control in the Cloud

Heterogeneous cloud infrastructures make it difficult to have effective controls to check privacy and other compliance constraints in an automated way. Cloud consumers have no means to verify that their policy requirements are being fulfilled.

Automated assurance is necessary to quickly evaluate the evidence that obligations with respect to personal data handling and business compliance requirements are being carried out (for instance, the collection of events showing who created a piece of data, who modified it and how, and so on). Governance, Risk management and Compliance (GRC) frameworks (e.g. SAP GRC[1]) are a common means of automating compliance in enterprises but do not provide much breadth or strong co-design of technical and legal mechanisms and although they can target specific regulations, they rarely deal with concepts like privacy and transparency, with the notable exception of recent work within CSA GRC Stack[2].

The accountability and privacy enforcement tool will enable enforcement of policies in ways that can be verified and audited. It will comprise a secure architecture facilitating verification of security and privacy requirements by external auditors as well as can

[1] `http://scn.sap.com/community/grc`, last visited on 22/11/2013

[2] `https://cloudsecurityalliance.org/research/grc-stack/`, last visited on 22/11/2013

provide notifications to end users. It receives as input machine-readable representations of policies in the accountability policy language, for instance, A-PPL, and takes a data-centric approach for the usage, monitoring, and obligation enforcement.

3.2 Introduction to Data Transfer Control Toolkit

Figure 3.1 presents the architecture for accountable personal data tracking in the cloud. The architecture is generic and can be used not only for monitoring personal data transfers, but also potentially any sensitive data. This may include business sensitive data (e.g. financial records, product designs, intellectual property, etc). At each service layer, we add a reference monitor, interacting with the policy engine components (illustrated in Figure 2.1). The accountability service (AS) interacts with the reference monitors to provide the interface with different users (Cloud Consumer, external auditors or Cloud Service Provider) The information collected by the reference monitors is then processed by AS, from which auditors and Data Controllers can check compliance to data privacy regulations, contracts and security standards. AS will manage the data segregation for multiple tenants by using cryptographic means, this is further discussed in Section 3.10.

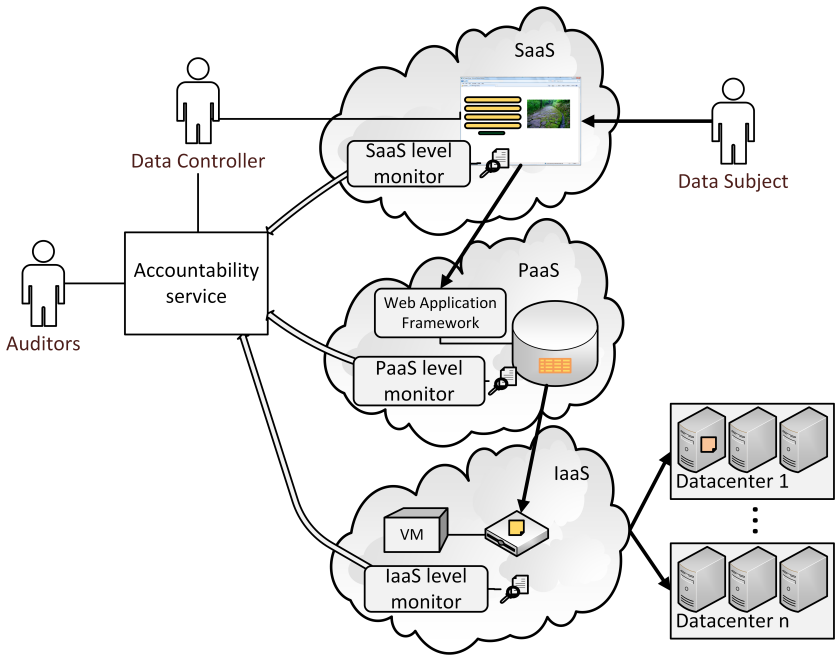

Figure 3.1: Data tracking architecture

Effective and profitable utilization of cloud services relies on data transfer and storage across services and different cloud infrastructures (which may have different jurisdictional restrictions). An open problem is how to find a balance between data provenance and related privacy or other regulatory constraints in the cloud, where physical perimeters are not clearly delimited. The lack of tools to support data localization and

transfer across services and cloud infrastructures creates barriers to cross-border considerations and different jurisdictional restrictions. Incompatibilities between jurisdictions affect privacy assurance. Even within the EU, regulatory requirements are defined at a national level and can differ.

The Data Transfer Control Toolkit addresses the lack of tools to support accountable data localization and transfer across cloud software, platform and infrastructure services, usually run by data processors. We designed a framework for automating the collection of evidence that obligations with respect to personal data handling are being carried out in what concerns personal data transfers.

3.3 Data Tracking Challenges in the Cloud

In this section we explain the challenges in data tracking in the cloud.

In order to effectively enforce privacy obligations the data should be tracked at all levels in the cloud ecosystem (application, platform, infrastructure). Currently the relationships between the virtual and physical data locations are not transparent to the cloud consumers (Data Subjects, Data Controllers) [KLP11], thus creating a serious barrier, which is preventing the providers from achieving accountability. The virtualization, employed in cloud computing, creates a level of indirection that obscures the physical location of the data in the cloud. This mapping is performed by a IaaS solution and is usually dynamic and opaque to the cloud users. Physical borders of the cloud infrastructures are also vague, hence often Data Controllers are not aware of where the data resides (at which Data Processor, in what country).

There are two types of transfers: the data can be transferred either vertically, that is from one service layer to another, or horizontally, i.e. in a given layer, data is moved due to elasticity, load balancing, backup, etc. Examples of horizontal transfers are: VM migration to another host (infrastructure level), migration of a tenant to another database server instance (platform level), as well as replications performed by the CSP for enabling disaster recovery, that are usually unknown to the cloud consumer (software, platform and infrastructure levels); examples of vertical transfers are: storing Data Subject's personal data in a database record, flushing a database record to disc. During data transfers in the cloud, the association between the data representation at a given service level, its sensitivity, the responsible Data Controller and related obligations are difficult to map. Monitoring both vertical and horizontal data transfers and maintaining this association to the Data Controller's obligations is essential to ensure accountability.

3.4 Data Tracking Monitors

In this section we describe our architecture for monitoring data transfers in the cloud that utilizes Data Tracking Monitors[3].

[3]please note that this name maybe a subject to change in the future versions to avoid the confusion with Data Track tool that is also planned to be delivered in the scope of A4Cloud project and that is aiming to provide the PII tracking visualization specifically for Data Subjects

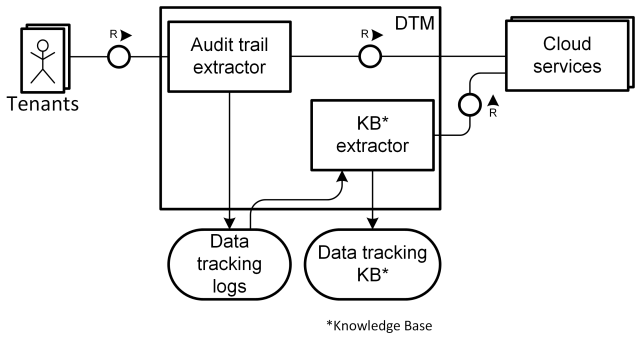

Figure 3.2: Data Tracking Monitor Architecture

Figure 3.2 illustrates the DTM architecture. In this approach DTM has a proxy that monitors all API calls from tenants (e.g. Data Controller or cloud platform administrator) to the cloud services, and extracts the audit trail in the format defined in Sec. 3.9. The DTM can optionally query the cloud services for some additional information. Further, the audit trail is used to construct data tracking knowledge base that represents operations on personal data as logical facts, suitable for automated analysis by the AS.

3.5 SaaS and PaaS Monitoring

We show some examples of operations at the platform level influencing on data transfers. In our use case, a database server instance is shared by multiple tenants at the application (SaaS) level. We associate the tenant id to the Data Controller and the corresponding data subject's personal data set identifier. This can be done at the moment of the tenant creation, for instance. Below we present some examples inspired from the SAP HANA Cloud Platform API [SAP13]. For database tenant creation, the RESTful PaaS level API calls observe the following schema:

URL	https://(host)/persistence/admin/tenants/(name)
Method	PUT
Returns	201 Created and tenant JSON
	400 Bad Request
	403 Forbidden
	500 Internal Server Error

A concrete example in JSON format would be as follows:

```
https://paasport.com/persistance/admin/MarcheAzur/fidelity
HTTP/1.1 200 OK
X-Compute-Request-Id: req-c461e07-9b18-4d9c-898b-37262fd9063
Content-Type: application/json
Content-Length: 1420
Request Method: PUT
```

```
Date: Mon, 27 May 2013 07:42:02 GMT

{
"create" : {
"name" : "MarcheAzur",
"database" : "paasport_instance_1",
"creation_time" : "2013-07-16 3:11 pm",
"creator" : "pass_provider_db_admin",
"dbuser" : "db_admin",
"password" : "admin1234"
}
}
```

The response for this request is the following:

```
{ "database_space": {
        "id": "34465727-8c79-22a0-6b24-567f565bc83c",
        "tenant_id": "MarcheAzurFR",
        "creator" : "paas_provider_db_admin",
        "dbuser" : " db_admin",
        "name": "fidelity_program",
        "created": "2013-07-16T15:11:10Z",
        "hostId": "e4d909c290d0fb1ca068ffaddf22cbd0",
        "jdbcconnection":
           "jdbc:sap:dbdriver://dbhost1.paasport.com:2323",
}
```

A second example of operation monitored at the PaaS level is to migrate a tenant database to another database server instance with distinct performance:

```
https://ma.paasport.com/persistance/admin/MarcheAzur/fidelity
HTTP/1.1 200 OK
X-Compute-Request-Id: req-c41e07-9b18-4d9c-898b-3762fd9063
Content-Type: application/json
Content-Length: 1420
Request Method: PUT
Date: Mon, 27 May 2013 07:42:02 GMT

{       "migrate_tenant" : {
        "name" : "MarcheAzur",
        "database" : "paasport_instance_2",
        "creation_time" : "2013-07-17 00:11 am",
        "creator" : "pass_provider_db_admin",
        "dbuser" : " db_admin",
        "password" : "admin1234"
        }
}
```

Whose response would consist in the following:

```
{       "database_space": {
        "id": "34465727-8c79-22a0-6b24-567f565bc83c",
        "tenant_id": "MarcheAzurFR",
        "creator" : "pass_provider_db_admin",
        "dbuser" : " db_admin",
        "name": "MarcheAzur",
        "created": "2013-07-16 T00:15:10Z",
        "hostId": "e4d909c290d0fb1ca068ffaddf22cbd0",
        "jdbcconnection":
           "jdbc:sap:dbdriver://dbhost2.paasport.com:2323",
}
```

In summary, many operations at the platform level can have implications to data transfers. The strength of our approach is to provide mappings from a service delivery layer to the underlying ones.

At the SaaS level, APIs are application specific. The SaaS provider is then responsible to adopt a privacy management approach and to identify which sensitive parts of the business process can transfer personal data from and to external systems, possibly under the control of third parties, and subject to regulatory constraints. Specific privacy concerns for the use case described here exist, some of them are mentioned in [YSSdO12].

3.6 IaaS Monitoring

At the infrastructure level, we monitor read-write accesses from upper layers or transfers occurring within the infrastructure, or between two IaaS providers. In this work we analyzed relevant operations provided by the OpenStack API [Ope]. OpenStack is an open source cloud solution that provides compute, network and storage services. Any operation required by a tenant or an admin user is performed through the API.

In OpenStack, we distinguish three types of entities that can hold data: instances, volumes and object stores. Instances are virtual servers managed by the compute service. They can store data in their file system. Volumes are block storage entities. A tenant can create a volume of the desired size and attach it to an instance. A volume can only be attached to a single instance at a time, but can be detached and attached to any instance. It can be seen as an external hard drive being plugged to a virtual machine. Object store is a persistent storage for static data. Creation, configuration and deletion of these entities can be detected by monitoring the API calls to the services that manage them.

Data can then be transferred within the infrastructure due to mechanisms such as snapshot, replication, instance migration, etc. For instance, let us assume there is a virtual machine, in which is stored a file with the mail addresses of the data subjects. When a snapshot of the instance is required, the mail addresses will be copied and included in the snapshot. The snapshot is managed by the OpenStack image service, thus will be stored in the host that owns the service. From this point, the tenant can create several virtual machines using this snapshot. Therefore, the mail addresses will also be stored in the new instances that are booted on the snapshot image. When the data is stored in a volume, it will also be duplicated when a snapshot of the volume is requested. Then, a new instance can boot on the volume snapshot, or a new volume can be created from this snapshot, resulting in a data duplication in both cases. At the volume creation, a scheduler defines the storage node on which it will be located. The OpenStack Object Storage service can be used to store backups of volumes and instances. The objects stored in OpenStack are replicated to ensure availability. Thus, a file will be copied at least on three disks, preferably located in different availability zones in the OpenStack cloud. For load-balancing purposes, a data replicate might be moved from one place to another. Due to load balancing operations or after a (physical) host

failure, an instance might be migrated from one server to another. All the operations described above result in duplication of data, where the data duplicate could either be located in the same region or country, or in a different one.

Data transfers can also occur between two IaaS providers. Indeed, one could use a cloud infrastructure for computing services and another one for storage services. We can again use the OpenStack example, that has a compatible API to the Amazon S3 API. Thus, we could easily imagine a scenario involving OpenStack and Amazon clouds together.

In an OpenStack environment, the DTM monitors all the API calls to log the events and to fill the data tracking knowledge base. It also has to send API requests to the OpenStack services in order to get the details about the servers that are not present in the monitored API calls. In particular, if the tenant requests a VM migration, the new host is not given in the response message. Moreover, some information are not given to a normal tenant, while it can be obtained if the tenant has an admin role. We assume our tool would get the same level of information as an admin tenant.

The Knowledge Base Extractor uses a logical knowledge engine called PyKE [Fre08], which allows performing logic programming in Python. Logic programming lends itself well to this case, since it makes it straightforward to perform pattern matching on messages, and to generate decisions based on available facts. The following code excerpt illustrates facts from the data tracking knowledge base:

```
#Volume Snapshot creation request
#volume_snapshot_creation(name, description, volume_id)

volume_snapshot_creation_req(database_volume_replication,
volume_Snapshot_for_PaaSPort_DB_Duplication,
e26a64348a5040458a524d5fff7d313e)
```

These facts showing the volume snapshot creation are built from the original request given below, intercepted by the DTM:

```
POST /v1/93859cf00e7741d4bdb6a37285cc827a/snapshots HTTP/1.1
        Host: 10.55.129.42:8776
        Content-Length: 145
        x-auth-project-id: admin
        accept-encoding: gzip, deflate
        accept: application/json
        x-auth-token: 0d608bb2235040ffb557fef2bbee826d
        user-agent: python-cinderclient
        content-type: application/json

        {"snapshot": {"display_name":
           "volume_Snapshot_for_PaaSPort_DB_Duplication",
        "force": "True", "display_description": null,
        "volume_id": "e26a6434-8a50-4045-8a52-4d5fff7d313e"}}
```

The HTTP response contains the following information:

```
HTTP/1.1 200 OK
        X-Compute-Request-Id:
          req-2260d35f-6503-4aa2-aa35-7a1a1558c5b0
        Content-Type: application/json
        Content-Length: 251
        Date: Tue, 25 Jun 2013 06:57:00 GMT
```

```
{"snapshot": {"status": "creating",
"display_name":
  "volume_Snapshot_for_PaaSPort_DB_Duplication",
"created_at": "2013-06-25T06:57:00.463182",
"display_description": null,
"volume_id": "e26a6434-8a50-4045-8a52-4d5fff7d313e",
"id": "0e55163e-794f-4712-87d6-ec10e9070941",
"size": 1}}
```

In our running example, this corresponds to creating a copy of the volume containing the PaaSPort database server instance. Below we show part of the rule set for tracking volume duplication and attachment to server instances in OpenStack. These capture the fact that volume snapshots can be used to boot server instances. The `holds_pii` predicate is used to "tag" an infrastructure object as containing personal data for a given data subject group (`data_subjects_id`):

```
personal_data_propagation_volume
foreach
  iaas_level_rules.holds_pii($instance_id, $data_subjects_id)
  iaas_level_rules.os_volume_attach($instance_id, $volume_id)
assert
  iaas_level_rules.holds_pii($volume_id, $data_subjects_id)
personal_data_propagation_vol_snapshot
foreach
  iaas_level_rules.volume_snapshot_creation($name,
    $description, $volume_id)
  iaas_level_rules.volume_snapshot_creation_resp($name,
       $description, $volume_id, $snapshot_id )
  iaas_level_rules.holds_pii($volume_id, $data_subjects_id)
assert
  iaas_level_rules.holds_pii($snapshot_id, $data_subjects_id)
personal_data_propagation_vol_creation
foreach
  iaas_level_rules.create_volume_req($snapshot_id, $name,
    $availability_zone, $attach_status, $project_id)
  iaas_level_rules.holds_pii($snapshot_id, $data_subjects_id)
  iaas_level_rules.create_volume_resp($status,
    $display_name, $snapshot_id, $volume_id)
assert
  iaas_level_rules.holds_pii($volume_id, $data_subjects_id)
personal_data_propagation_vol_attach
foreach
  iaas_level_rules.os_volume_attach($instance_id, $volume_id)
  iaas_level_rules.holds_pii($volume_id, $data_subjects_id)
assert
  iaas_level_rules.holds_pii($instance_id, $data_subjects_id)
```

3.7 Topology

In order for the AS to get the physical location of data related to a specific Data Controller, we introduce a topology knowledge base that captures the mapping between the virtual machines, images and volumes on one side and their physical representation (availability zones, network, host) on the other side.

In particular, it describes the physical boundaries of the infrastructure, the mapping between virtual servers and physical hosts and the location of the hosts. The mapping

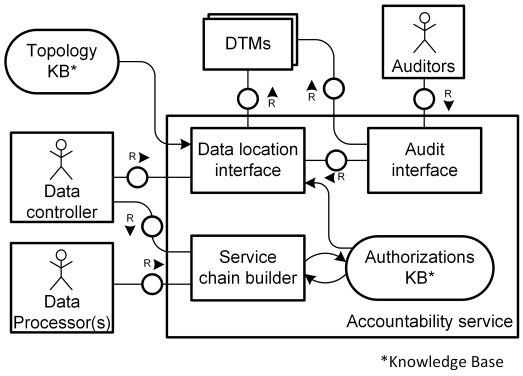

Figure 3.3: Accountability service architecture

is provided and managed by the infrastructure provider, in contrast to AS, and hence its trustworthiness is linked to that of the CSP. As data processor, the the IaaS provider is responsible for creating and maintaining this information up to date, as it is the only entity aware of the constantly changing physical landscape of the cloud infrastructure. We estimate that the cost of maintaining this information is negligible, as network administration practices already require to keep this kind of record. The following code excerpt illustrates facts contained in this knowledge base:

```
#host(host_name, host_id)
host(Infrared_IE, e9ef8cf20d89f8ee6cfa6)

#Determines where a host is located
#host_location(host_name, location)
host_location(Infrared_IE, Ireland)

#Instances for a given host
# instance_host(host_id, instance_id)
instance_host(e9ef8cf20d89f8ee6cfa6,
  a4ab4ef7c92264d1c8c14958f8d6f4318)
```

The excerpt above shows the mappings among data processors and the corresponding data handling objects at the infrastructure level. Additionally, it also relates the hosts maintained by the infrastructure provider to their actual physical location.

3.8 Accountability Service

Figure 3.3 illustrates the architecture of the AS. The main component in AS is Data location interface, that enables Data Controllers and auditors to determine the location of the Data Subjects' data entrusted to the Data Controller: both logically (i.e the virtual hosts which store the data) and physically, because the geographic location of the servers is known. In particular, it returns all Data Processors holding the Data Subjects' data, the sensitivity category of personal data and the physical storage location at the country level. This allows Data Controllers to check their state of compliance with the privacy obligations: they could determine potential unauthorized transfers of personal data to

other parties and storage in a country, which is not considered as offering sufficient data privacy protection guarantees.

In order to provide this response the Data location interface aggregates information from all the DTMs — more precisely from data tracking knowledge bases — in the cloud ecosystem of this particular Data Controller. In addition, it makes use of the Topology KB (see 3.7) in order to derive the physical location of the data; and Authorizations KB, that captures the data processors and other third parties that are authorized by Data Controller to process its Data Subjects' personal data. This database is filled together by the Data Controller and Data Processors (Data Processors can further delegate the processing to other parties with the prior permission of Data Controller and consent from the data subjects). Thus, if there is a transfer of the personal data to a party outside this list, it is deemed as a violation of privacy obligations.

The following code snippet shows possible facts in Authorizations KB:

```
data_subjects(1000, Europe)
data_controller(MarcheAzur, 1000, France, SaaS)
data_processor(PaaSPort, Germany, PaaS)
data_processor(Infrared, Ireland, IaaS)
authorized_party_transfers(MarcheAzur, PaaSPort, 1000)
authorized_party_transfers(PaaSPort, Infrared,1000)
authorized_location_transfers(Europe, Germany, 1000)
```

The following code shows a forward chaining query to identify the multiple locations the data for a given data subject set:

```
personal_data_location
 foreach
  iaas_level_rules.instance_host($host_id, $instance_id)
  iaas_level_rules.host($host_name, $host_id)
  iaas_level_rules.host_location($host_name, $location)
  iaas_level_rules.holds_pii($instance_id, $data_subjects_id)
 assert
  iaas_level_rules.personal_data_location($data_subjects_id,
   $location)

with fc_goal.prove(engine,
 data_subjects_id1=data_subjects_id1) as gen:
 for vars, plan in gen:
  print "%s is located in %s" % \
   (data_subjects_id1, vars['location1'])
```

Auditors could check all operations performed on personal data and verify if there were any possible violations with respect to data transfers. This also relates to the Cloud Trust Protocol, as it provides a way to respond to the Element of Transparency number 8 which requests disclosure about geographic location of "units (including data sets) being used on behalf of the consumer"[KE10].

3.9 Audit Trails

In order to enable the correlation between the logs from different service levels coming from various sources, we define a common log entry structure, adapted from [KLP11]:

$\langle Actor, Operation, When, Where \rangle$

Where:

- $\langle Actor \rangle$ is the initiator of the operation and can be the tenant id, CSP operator name, etc.
- $\langle Operation \rangle$ is the action on the personal data item. The actual operations are dependent on the implementation and on the service level. In order to normalize these differences we distinguish the following operation categories as the one that are related to data tracking: Create, Read, Update, Delete, Copy — which are mapped to actual operations.
- $\langle Data \rangle$ is the identifier of the data object, under which the operation is performed. It is linked with a personal data category and Data Controller(see Section 3.4).
- $\langle When \rangle$ is the event date and time.
- $\langle Where \rangle$ is the data processing location. Depending on the service level this can be a database server instance (platform level), a virtual disk volume, a physical host IP or the geographic location (infrastructure level).

Depending on the service level these elements differ and are described in the following sections. Logs coming from different sources are normalized according to this format, translated into logical facts and stored in the DTM knowledge base (detailed below).

3.10 Security Considerations

Here we describe some preliminary analysis of possible threats to the solution and propose some mitigations. We will provide a detailed attacker model in the next phases of the work on the D3 work package in order to design "tamper proof" policy enforcement mechanisms. The main assumption in this work is that the CSPs are trusted and they would configure their service so that all control operations flow through DTMs. Otherwise CSPs would fail to provide transparency with respect to the location of the data processing, ultimately losing market share.

An adversary tampers with the logs or knowledge bases generated by a DTM Our solution has to protect the integrity of the logs. The principle of *forward security* [SK98, MT09, YN09] can be applied to the logs, that is an attacker must not be able to undetectably modify or delete log entries that were generated before the compromise. Integrity of logs or data tracking knowledge bases can be protected by adding a Message Authentication Code (MAC). In this case, the DTM could sign all log entries with a private key, provided e.g. by a Trusted Platform Module (TPM)[4] and Auditors could verify their integrity. We also consider that our DTMs will consume trusted timestamps and that further configuration data have also been certified by audits to minimize this risk.

[4] `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`

An adversary steals the logs generated by a DTM Audit trails generated by DTMs (see Sec. 3.9) can contain sensitive personal data. Thus, the confidentiality of data tracking logs has to be protected. Confidentiality of the logs can be obtained through a combination of symmetric and asymmetric encryption (for performance reasons), where the private key is shared between the Data Controller and Auditors. Moreover, to ensure segregation between the different Data Controllers, the encryption keys will be specific to each Data Controller, as he is liable for the personal data location. Particular situations may demand to extend keys to data processors if contractual provisions are made, or also in the case of joint Data Controllers (also called co-controllers).

An adversary tampers with the DTM As the DTMs are located in the CSP's premises (for performance reasons) they are specifically vulnerable to tampering. In order to ensure the generation of evidence by DTMs is not tampered with, apart from proper physical and logical access control, they could use TPMs for remote attestation of DTM's configuration.

CSP is intentionally bypassing DTM If a CSP uses a covert channel instead of the service APIs, potential violations can happen without being detected. This risk is somewhat diminished in case of frequent on-site audits, but should still be taken into account. Our DTM can be incremented with additional rules to identify such data transfers by observing events are the OS level and at the networking interfaces.

A malicious software makes transfers to unauthorized locations If the software running on CSP premises, e.g. a SaaS solution, is provided by a third party, there is a threat of it sending data to locations without the consent of other parties. For this, it may intentionally use covert channels instead of common API monitored by DTMs. This could be mitigated by monitoring not only the normal API calls but all the outside communication at the network level, using e.g. an application-level gateway. The logs of this gateway could also be collected by AS to enable later inspections.

Data protection authorities abuse of data tracking The PRISM scandal[5] showed that the threat of a State spying on the communications between civilians in the Cloud is real. Any solution, including this one, allowing the authorities to track all data movements should be assessed against the threat of possible abuse. The ideal approach would be to have completely anonymized logs, at the same time capturing other important information (responsible Data Controller, data sensitivity, location, etc.).

The design and experimental implementation of the assurance mechanisms to protect the data generated and handled by the Data Transfer Control Toolkit will be the subject of the coming tasks in scope of this work package, along with other improvements.

[5] `http://www.theguardian.com/world/prism`

Conclusions

The main goal of this document is to pave the way towards implementation of the tools for policy enforcement and compliance. We have presented the need for the additional tooling related to the accountability policies (AccLab) as well as the candidate technology for a work related to policy enforcement (PPL Engine).

AccLab will be further developed in scope of the task that is aimed to deliver accountability and privacy enforcement tools. We have provided overview of the architecture and positioned the tool in the overall policy enforcement framework. As the work related to the enforcement methodologies continues in the scope of WP C-4, we are planning to integrate the results of that work in the agile manner in our ongoing development process.

After overview of the current state of the policy enforcement tools for PPL, we have established a feasibility of supporting the extensions envisioned in the A-PPL, introduced by the A4Cloud policy framework. The extensions related to logging, audit and evidence collection are of major focus when accountability is concerned. Thus we have planned to further investigate the progress related to the techniques planned to be delivered by WP C-4. However in case of the actual policy enforcement, we can already envision that the primary focus will be to adapt the existing PPL Engine to support the new language constructs. After that step, the additional mechanisms designed and described in this report, can be included to support each of the extended language feature.

The complementary enforcement framework related to the sticky policies will be also one of the major focus for the implementation task. In this document we have positioned the tool in the overall enforcement architecture, however a detailed architecture is still a work in progress.

In parallel with the accountability and privacy enforcement tools, the effort in this work package is split on data transfer control toolkit. This document provides a working example of such tool, with a detailed architecture view, as well as a use-case, selected among the three use-cases produced by WP B-3 [BOS+13]. As the tool itself presents quite complete approach for the data tracking in the cloud ecosystem, we may further work on generalizing the approach and removing some of the constraints (e.g., related to the necessary cloud landscape adaptation) as well as the performance related concerns (processing large amount of log data produced by a cloud landscape). The implementation work on this tool will continue in the scope of the task aimed at developing tools to control data transfer in the cloud.

As the policy enforcement tools are just one group of the software components building up overall A4Cloud tools architecture, we need to further investigate how the mechanisms related to the policy enforcement will interface with the external components. Such composition of the tools is briefly mentioned in the sections related, e.g., to evidence collection. Also overall AccLab architecture gives a possibility for both input (e.g., contracts) and output (concrete enforcement mechanisms) flows of data and messages between the tools delivered by the other work packages.

As the secondary objective for this report we are contributing to the A4Cloud ref-

erence architecture. We have documented the envisioned architecture of the tools, as well as introduced the interfaces where such tools can be further extended or combined with the other A4Cloud tool chain. We plan later on to analyze the cross-work-package concerns related to the integrated accountability tools landscape. A dedicated effort in WP D-2 (”Reference Architecture”) will help to adapt the tools for the policy enforcement in order to deliver interoperable toolkit for accountability in the cloud.

Appendix A

Enforcement Methodologies Survey

The development of the enforcement methodologies has been recently initiated in WP C-4 ("Policy mapping and representation"). We report here our first investigations on this subject, performed as a preliminary study established at the initial phase of the work package.

We first review some related work for policy enforcement and more specifically in a cloud context. There are currently several proposals related to enforcement policies in the cloud but mainly for security policies. These proposals discussed tools, frameworks and methodologies but mainly technical solutions related to access control or more broadly usage control. An holistic view of accountability from legal and regulatory obligations to software enforcement is generally lacking.

One may note the survey in [FJWX12] about accountability. It evaluates several approaches and it proposes to consider three aspects: time/goals, information, and action for accountability systems. The time/goals identifies five steps:

- Prevention, this step is concerned with managing security issues.
- Detection, this step is related to the detection of a violation.
- Evidence, this step gathers or stores evidences about a violation.
- Judgment, this step identifies the violator of a policy and more generally, all the persons involved in it.
- Punishment, this step is concerned with remediation and sanction actions.

The information aspect is related to privacy and not related to the enforcement question. Action aspect is related to some collaboration issues and the processing of the previous steps. The time steps give a general flow which can be valuable to understand and design an accountability framework.

In [HKK12] the authors identify several challenges for information assurance in the cloud: the semantic diversity of data, the internal structure of the cloud system and the

accountability policies. The semantic diversity of data implies that policies should apply to a wide range of data from structured to unstructured. Access control granularity and information processing can be different. A general framework for policy enforcement should consider three dimensions: (1) data type (e.g., relational data, RDF data, text data, etc.), (2) computation (e.g., SQL queries, SPARQL, Map/Reduce tasks, etc.), and (3) policy requirements (e.g., access control policies, data sharing policies, privacy policies, etc.). The authors propose a framework supporting context-based access control and privacy preservation rules. For the enforcement they rely on in-lined reference monitors and a trusted application programming interface for cloud system.

[BKF+11] presents NetODESSA, an inference-based system for network configuration and dynamic policy enforcement. NetODESSA extends the ODESSA distributed which is a dynamic policy monitoring system. The system allows network administrators to write policies enforced at the network-level and monitored at the host-level. This approach is based on Resource Description Framework (RDF), OpenFlow and NOX. The main idea is to write some abstract high-level policies in term of host characteristics. Then an inference engine, using data observed from the network, is able to classify the hosts and thus to determine if violations occur and to take a proper action.

A general view of the accountability life-cycle and the challenges is addresses in [KLP11]. The accountability life cycle is compound of seven steps in a way similar to an iterative software lifecyle. These steps are: Policy planning, sens and trace, logging, safe keeping of logs, reporting and replaying, auditing, optimizing and recycling. The second proposition relates to the question of the nature of data to log. The principle is to consider three layers: the workflow, the data and the system layers. The authors consider three technical approaches to increase accountability: Central Watchdog/ Manager Service, Local File Tracking Embedment, and Domain segregation.

One piece of work related to usage control policy enforcement is [HBP05, HPB+07]. This work proposes a general analysis of obligations and a formal language for expressing usage control. A discussion is made about enforcement in [HBP05] but it focuses on a central monitor. In [PHS+08] the authors focus on mechanism inspired by DRM system, that is observation mechanism. Such a mechanism has one provider-side that monitors the consumer-side. The consumer-side signals to the provider-side what are the actions it is being processed.

The work of Yumerefendi and Chase [YC04] is a first attempt to define a framework for accountability. They consider that: ”Accountable systems expose interfaces that allow them to verify the correctness of their actions.” This is not an enforcement tool support but it is a general framework which has some interesting features. The focus is to enable accountability for services that access and update internal state in response to invocations from clients. The foundation of the approach relies on digitally preserving signed records of actions and internal state snapshots of each service. These kind of secure logs are then used to detect tampering, or to prove responsibility for unexpected states or actions. A more focused and concrete approach by the same authors is [YC07]. They presents the design and implementation of a network storage service. The CATS system supports strong accountability and a critical evaluation of its performances. This is a simple web services interface that allows clients to read and write

opaque objects of variable size. However, regarding policy enforcement the proposal is far from to be generic since the policy to use the create, read and write actions are predefined by the system. Thus it is not flexible to define new user preferences and new service obligations.

Bibliography

[Ate] Giuseppe Ateniese. Verifiable encryption of digital signatures and applications. *ACM Trans. Inf. Syst. Secur. (TISSEC)*, (1):1–20.

[BKF+11] J. Bellessa, E. Kroske, R. Farivar, M. Montanari, K. Larson, and R.H. Campbell. Netodessa: Dynamic policy enforcement in cloud networks. In *30th IEEE Symposium on Reliable Distributed Systems Workshops (SRDSW)*, pages 57–61, 2011.

[BOS+13] Karin Bernsmed, Anderson Santana De Oliveira, Jakub Sendor, Nils Brede Moe, Thomas Rübsamen, Vasilis Tountopoulos, and Bushra Hasnain. D:B-3.1: Use Case Descriptions. Technical Report D:B-3.1, Accountability for Cloud and Future Internet Services - A4Cloud Project, June 2013.

[FJWX12] Joan Feigenbaum, Aaron D. Jaggard, Rebecca N. Wright, and Hongda Xiao. Systematizing "accountability" in computer science. Technical Report YALEU/DCS/TR-1452, University of Yale, 2012. www.cs.yale.edu/publications/techreports/tr1452.pdf.

[Fre08] Bruce Frederiksen. Applying expert system technology to code reuse with pyke. In *PyCon*, 2008.

[GdOS+13] Alexandr Garaga, Anderson Santana de Oliveira, Jakub Sendor, Monir Azraoui, Kaoutar Elkhiyaoui, Melek Önen Refik Molva, Ronan-Alexandre Cherrueau, Rémi Douence, Hervé Grall, Jean-Claude Royer, Mohamed Sellami, Mario Südholt (EMN), and Karin Bernsmed. D:C-4.1: Policy Representation Framework. Technical Report D:C-4.1, Accountability for Cloud and Future Internet Services - A4Cloud Project, 2013.

[HBP05] Manuel Hilty, David A. Basin, and Alexander Pretschner. On obligations. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, volume 3679 of *Lecture Notes in Computer Science*, pages 98–117. Springer, 2005.

[HER] HERAS AF team. HERAS AF (Holistic Enterprise-Ready Application Security Architecture Framework). `http://herasaf.org/`.

[HKK12] Kevin W. Hamlen, Lalana Kagal, and Murat Kantarcioglu. Policy enforcement framework for cloud data management. *IEEE Data Eng. Bull*, 35(4):39–45, 2012.

[HPB+07] Manuel Hilty, Alexander Pretschner, David A. Basin, Christian Schaefer, and Thomas Walter. A policy language for distributed usage control. In Joachim Biskup and Javier Lopez, editors, *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 531–546. Springer, 2007.

[KE10] Ronald Knode and Douglas Egan. Digital trust in the cloud: Into the cloud with ctp — a precis for the cloudtrust protocol, v2.0. `https://cloudsecurityalliance.org/wp-content/uploads/2011/05/cloudtrustprotocolprecis_073010.pdf`, 2010. Cloud Security Alliance.

[KLP11] Ryan K. L. Ko, Bu-Sung Lee, and Siani Pearson. Towards achieving accountability, auditability and trust in cloud computing. In Ajith Abraham, Jaime Lloret Mauri, John F. Buford, Junichi Suzuki, and Sabu M. Thampi, editors, *Advances in Computing and Communications - First International Conference, ACC 2011, Kochi, India, July 22-24, 2011, Proceedings, Part IV*, volume 193 of *Communications in Computer and Information Science*, pages 432–444. Springer, 2011.

[MT09] Di Ma and Gene Tsudik. A new approach to secure logging. *TOS*, 5(1), 2009.

[Ope] OpenStack. OpenStack API reference. `http://api.openstack.org/api-ref.html`. [Online, accessed 11-July-2013].

[PHS+08] Alexander Pretschner, Manuel Hilty, Florian Schütz, Christian Schaefer, and Thomas Walter. Usage control enforcement: Present and future. *IEEE Security & Privacy*, 6(4):44–53, July/August 2008.

[Pri11] PrimeLife Consortium. PrimeLife. `http://primelife.ercim.eu/`, 2011.

[SAP13] SAP. SAP HANA cloud platform. `http://scn.sap.com/community/cloud-platform`, 2013.

[SK98] Bruce Schneier and John Kelsey. Cryptographic support for secure logs on untrusted machines. In *Proceedings of the 7th conference on USENIX Security Symposium - Volume 7*, SSYM'98, pages 4–4, Berkeley, CA, USA, 1998. USENIX Association.

[TNR11] Slim Trabelsi, Gregory Neven, and Dave Raggett. Report on design and implementation. `http://primelife.ercim.eu/images/stories/deliverables/d5.3.4-report_on_design_and_implementation-public.pdf`, 2011.

[YC04] Aydan R. Yumerefendi and Jeffrey S. Chase. Trust but verify: accountability for network services. In *Proceedings of the 11th workshop on ACM SIGOPS European workshop*, EW 11, New York, NY, USA, 2004. ACM.

[YC07] Aydan R. Yumerefendi and Jeffrey S. Chase. Strong accountability for network storage. *Trans. Storage*, 3(3), October 2007.

[YN09] Attila Altay Yavuz and Peng Ning. Baf: An efficient publicly verifiable secure audit logging scheme for distributed systems. In *ACSAC*, pages 219–228. IEEE Computer Society, 2009.

[YSSdO12] Peng Yu, Jakub Sendor, Gabriel Serme, and Anderson Santana de Oliveira. Automating privacy enforcement in cloud platforms. In Roberto Di Pietro, Javier Herranz, Ernesto Damiani, and Radu State, editors, *DPM/SETOP*, volume 7731 of *Lecture Notes in Computer Science*, pages 160–173. Springer, 2012.