

---

## D:C-8.3 Automation Service for the Framework of Evidence

---

<b>Deliverable Number</b>	D38.3
<b>Work Package</b>	WP 38
<b>Version</b>	Final
<b>Deliverable Lead Organisation</b>	HFU
<b>Dissemination Level</b>	PU
<b>Contractual Date of Delivery (release)</b>	31/03/2015
<b>Date of Delivery</b>	31/03/2015

---

### Editor

Christoph Reich, Thomas Rübsamen (HFU)

### Contributors

Tomasz Wiktor Włodarczyk, Rui Pais (UiS), Monir Azraoui, Melek Önen (EURECOM), Jenni Reuben, Tobias Pulls (KAU), Mohamed Sellami, Jean-Claude Royer (ARMINES---EMN), Massimo Felici (HP), Karin Bernsmed (SINTEF)

### Reviewers

Theo Koulouris (HP), Vasilis Tountopoulos (ATC)

## Executive Summary

The goal of the automation service for the framework of evidence is to provide services for evidence collection, processing, incident detection, and auditing in the A4Cloud toolkit. This is essential for the evidence-driven approach to demonstrate accountability in the cloud. The framework of evidence service has been implemented by the A4Cloud Tool: Audit Agent System (AAS). The term Audit Agent System stands synonymously for framework of evidence implementation.

This document is part of the two final deliverables in the WP-C8. The first deliverable is the final framework of evidence delivered by the Task 8.5. The deliverables of WP-C8 Task 8.6 are: (i) this document, which focuses on the technical and implementation aspects of the framework of evidence, the system of evidence collection and (ii) the actual A4Cloud tool Audit Agent System.

This document includes:

- A mapping of the principles and mechanisms defined in the framework of evidence to the implementation in the Audit Agent System
- An architecture for scalable automated evidence collection, monitoring, evidence storage and analysis/audit for supporting the provision of an account
- Integration of concepts from WP-C4: “Policy Mapping and Representation” in form of utilizing the A-PPL policy language as an important input for the Audit Agent System
- Integration in WP-D7: Instantiation for Use Case, to drive the development of collection and detection mechanisms in the Audit Agent system, by focussing on the demonstration scenarios
- An analysis of the scalability properties of the framework of evidence and its implementation the Audit Agent System, with the focus on potential bottlenecks in evidence collection and processing and possible architectural solutions
- An analysis of the privacy properties of the Audit Agent System, to enforce good data protection practices in the Audit Agent System itself

Furthermore, we have implemented the Audit Agent System and all the components described in this deliverable as a RESTful web service using software agent technology and a user interface based on Bootstrap in cooperation with WP-D5: User-centric Tools for Accountability.

## Table of Contents

Executive Summary.....	2
1 Introduction.....	5
1.1 Purpose.....	5
1.2 Glossary of Acronyms / Abbreviations.....	5
2 Related Work.....	7
2.1 Tools Supporting Evidence Industrial Practices.....	7
2.1.1 Preventive Mechanisms.....	9
2.1.2 Detective Mechanisms.....	10
2.1.3 Corrective Mechanisms.....	12
2.2 Frameworks of Evidence Collection.....	12
2.3 Scalable Monitoring, Event Processing and Analysis.....	13
3 Integration of Framework of Evidence by Audit Agent System.....	15
4 Scenario.....	17
4.1 Demonstration Deployment.....	17
4.1.1 <i>CardioMon</i> Audit Agent System (AAS).....	18
4.1.2 <i>Map-on-Web</i> Audit Agent System (AAS).....	18
4.1.3 <i>DataSpacer</i> Audit Agent System (AAS).....	19
4.2 Scenario A: A-PPL-based.....	19
4.2.1 Scenario A1: Data Retention.....	19
4.2.2 Scenario A2: Notification.....	20
4.2.3 Scenario A3: Right to Know vs. Need to Know.....	20
4.3 Scenario B: Incidents.....	21
4.3.1 Scenario B1:SSL Scare.....	21
4.3.2 Scenario B2: Service Availability.....	22
5 Evidence Collection Service.....	23
5.1 Architecture of the Audit Agent System (AAS).....	23
5.1.1 <i>Input</i> : Audit Policy Module (APM).....	23
5.1.2 <i>Runtime Management</i> : Audit Agent Controller (AAC).....	23
5.1.3 <i>Collection and Storage</i> : Evidence Collection Agents, Evidence Store.....	23
5.1.4 <i>Processing and Presentation</i> : Evidence Processor, Presenter.....	24
5.2 Proof of Retrievability (POR).....	27
5.2.1 StealthGuard.....	27
5.2.2 Integration of Proofs of Retrievability in the A-PPL Engine.....	28
5.2.3 Integration in AAS.....	29
6 Interaction with the A-PPL Policy Language.....	31
6.1 From A-PPL to Audit Task.....	31
6.2 A-PPL Access Control to AAS Mapping.....	31
6.3 A-PPL Data Handling to AAS Mapping.....	32
6.4 Other Types.....	33
6.5 Audit Task Example.....	33
7 Interfaces of the Evidence Collection Service.....	36
7.1 Internal Interfaces.....	36
7.1.1 A-PPL Engine Interfaces.....	36
7.1.2 DTMT Interfaces.....	37

7.2	External Interfaces .....	38
7.2.1	Multi-Provider Data Exchange.....	38
7.2.2	REST Interface .....	39
8	Scalability of Evidence Collection .....	41
8.1	Non-technical Aspects of Scalability .....	42
8.2	Technical Aspects of Scalability.....	42
8.2.1	<i>Setup</i> Phase .....	42
8.2.2	<i>Runtime</i> Phase .....	43
9	Privacy Analysis of the Audit Agent System .....	45
9.1	Threat model and Risks .....	45
9.2	Privacy Enhancing Technologies for AAS .....	45
10	Conclusions .....	47
11	References .....	48
12	Index of figures .....	52
13	Index of tables .....	53

## 1 Introduction

As the cloud grows and the evidence about its behaviour grows ever-larger, manual methods become untenable. The framework of evidence is one of several areas within the project where effort must be put into demonstrating that the accountability approach is indeed scalable, or has the potential of becoming so. This document describes an automated service implementing key parts of the framework of evidence to instantiate a proof-of-concept of the system for evidence collection. These key parts are **gathering, evaluating, incident detection, and auditing** accountability evidence in typical scenarios encountered in cloud services. Significant parts of automation are interaction with policy languages from C-4 and instantiated use-case from D-7. During this time bottlenecks in throughput will be identified and corrective measures devised.

To take into account the highly dynamic infrastructure of a cloud and to handle the huge amount of potential evidence information, the Framework of Evidence has been based on a software agent system, called Audit Agent System (AAS) (Ruebsamen & Reich, 2013). The Audit Agent System (AAS) enables the automated audit of multi-tenant and multi-layer cloud applications and cloud infrastructures for compliance with custom-defined policies. Evidence collection agents can be deployed at different cloud architectural layers (i.e., network, host, Hypervisor, IaaS, PaaS and SaaS) with the purpose of evidence collection, processing and incident detection. On demand the AAS can generate audit reports. In AAS, the evidence collection process also includes monitoring activity that allows the evaluation of the effectiveness of controls, which is usually an internal process.

An audit component is central to any approach to accountability. Remediation actions that have to be performed after some policy has been violated, for instance, often rely on fine-grained monitoring facilities and extensive analysis capabilities of the resulting evidence. The AAS tool provides suitable means for the runtime monitoring of Cloud applications and infrastructures, the verification of audit policies against the collected evidence, and the reporting of policy violations along with the evidence supporting it.

### 1.1 Purpose

The approach is focused on implementing the D:C-8.1 Framework of Evidence of the WP C-8 to continuously collect evidence driven by rules and obligations defined in WP C-4, B-3, D-7 and the conceptual framework defined in WP C-2. Outcomes of the framework of evidence implementation support demonstration of accounts developed in WP D6 and as described in the general architecture framework elaborated in WP C-2.

### 1.2 Glossary of Acronyms / Abbreviations

AA	Auditing Authority
AAC	Audit Agent Controller
AAS	Audit Agent System
ACL	Agent Communication Language
APM	Audit Policy Module
CMS	Cloud Management System
CSP	Cloud Service Provider
CTP	Cloud Trust Protocol
DEB	Digital Evidence Bag
DTMT	Data Transfer Monitoring Tool
ECC	Error Correcting Code
FoE	Framework of Evidence
HTTP	Hypertext Transport Protocol
IaaS	Infrastructure as a Service
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IMS	Identity Management System
JADE	Java Agent Development Environment
JRE	Java Runtime Environment
MTA	Mail Transport Agent

PaaS	Platform as a Service
PII	Public Identifiably Information
PKI	Public Key Infrastructure
PoR	Proof of Retrievability
REST	Representational State Transfer
SaaS	Software as a Service
SLA	Service Level Agreement
SSL	Secure Socket Layer
TL	Transparency Log
TLS	Transport Layer Security
TPA	Third Party Auditor
XACML	Extensible Access Control Mark-up Language

## 2 Related Work

The related work is split up into “Industry Practices/Related Tools”, tools that are collecting and processing evidence, into “Framework of Evidence Collection”, and work that has been done in “Scalable Monitoring, Event Processing and Analysis”.

### 2.1 Tools Supporting Evidence Industrial Practices

The accountability-based approach (defined in (Felici & Pearson, 2014)) involves the adoption of different accountability mechanisms (that is, “*diverse processes, non-technical mechanisms and tools that support accountability practices*” (Felici & Pearson, 2014)). This section focuses in particular on tools that support the gathering of evidence. In general, we can assume that any accountability mechanism (tool) would generate evidence, process evidence or both. Most of the tools associated with evidence collection and analysis are concerned with operational aspects of evidence (e.g. monitoring of critical events). However, accountability promotes implementation of practical mechanisms whereby legal requirements and guidance are translated into effective protection for data. Legislation and policies tend to apply at the data level, but the mechanisms can be at various levels, including the system level and data level. A toolbox of mechanisms could be provided for data controllers, to allow construction of custom-built solutions, whereby the controllers might tailor measures to their context (e.g. taking into account consideration of the systems involved, type of data and data flows). We can co-design legal mechanisms, procedures and technical mechanisms to support the accountability-based approach. We may integrate design elements to support: prospective (and proactive) accountability, using **preventive mechanisms**, continuous accountability of data governance using **detective mechanisms**, and retrospective (and reactive) accountability, using **corrective mechanisms**.

- **Preventive Mechanisms** – These can be used to mitigate the occurrence of an action for continuing or taking place at all (e.g. an access list that governs who may read or modify a file or database, or network and host firewalls that block all but allowable activity). The cloud is a special example of how businesses need to assess and manage risk better (Pearson & Yee, 2013). Preventive controls for cloud include risk analysis and decision support tools (for example, Privacy Impact Assessments), policy specification and enforcement (for example, machine readable policies, privacy-enhanced access control and obligations), trust assessment, obfuscation techniques and identity management.
- **Detective Mechanisms** – These are used to identify the occurrence of a privacy or security risk that goes against the privacy or security policies and procedures (for example, intrusion detection systems, policy-aware transaction logs, language frameworks and reasoning tools). Detective controls for the cloud include audit, tracking, reporting, and monitoring.
- **Corrective Mechanisms** – These (e.g. an incident management plan, dispute resolution) are used to fix an undesired event that has already occurred.

Preventive, detective and corrective mechanisms complement each other: a combination of these would ideally be required in order to provide accountability. Provision of accountability would not just be via procedural means, especially for cloud, which is such an automated and dynamic environment. Technology can play an important role in enhancing the solution (e.g. by enforcing policies). Procedural measures for accountability include determining the capabilities of Cloud Service Providers (CSPs) before selection, negotiating contracts and Service Level Agreements (SLAs), restricting the transfer of confidential data to CSPs and buying insurance. Organisations should also appoint a Data Protection Officer, regularly perform privacy impact assessments on new products and services, and put mechanisms in place to allow quick response to data subject access and deletion requests. Technical measures for accountability can include encryption for data security mitigation, privacy intermediaries and agents to help increase trust. We also need to be able to rely on infrastructure to maintain appropriate separations, enforce policies and report information accurately. It could be argued that the current proposal directive for data protection (Boyens, Paulsen, Moorthy, Bartol, & Shankles, 2013) places too much emphasis on remediation of problems (e.g. privacy breaches), and not enough on trying to get organisations to *do the right thing* for data protection in the first place.

Our approach involves the provision of hybrid accountability mechanisms via a combination of policies, regulatory and technical means. It is a co-regulation strategy based on a corporate responsibility model

underpinned primarily by contracts. This approach places the onus upon the data controller to take a more proactive approach to ensuring compliance, and encourages cloud service vendors and subcontractors to compete in providing services on the basis of evolving better privacy and security enhancing mechanisms and processes. We build upon the accountability definitions and model, discuss and extend these to include prospective effects, that is to say, proactive rather than just reactive measures. This is because the policies by which we are judging our actors are constantly changing, the context and technological environment is changing and privacy-related harms to individuals are not equal. It is necessary to provide mechanisms to determine liability in the event of a breach, but we also (from the point of view of the data controller) build in processes and reinforce good practices such that the liability does not arise in the first place. We suggest ways in which an organisation might take an accountability approach further in order to develop a reflexive privacy process that is not simply a static compliance mechanism but rather that involves an on-going process of data protection monitoring and review and improvement throughout the contractual chain. Figure 1 shows examples of mechanisms supporting accountability. The identified mechanisms map to the trusted services supporting accountability that are developed by the Cloud Accountability Project or that relate directly to those. Other services could be provided and would fit into this framework, such as incident management, identity management services and certification. We provide benefits and tools for a range of different stakeholders (Figure 1).

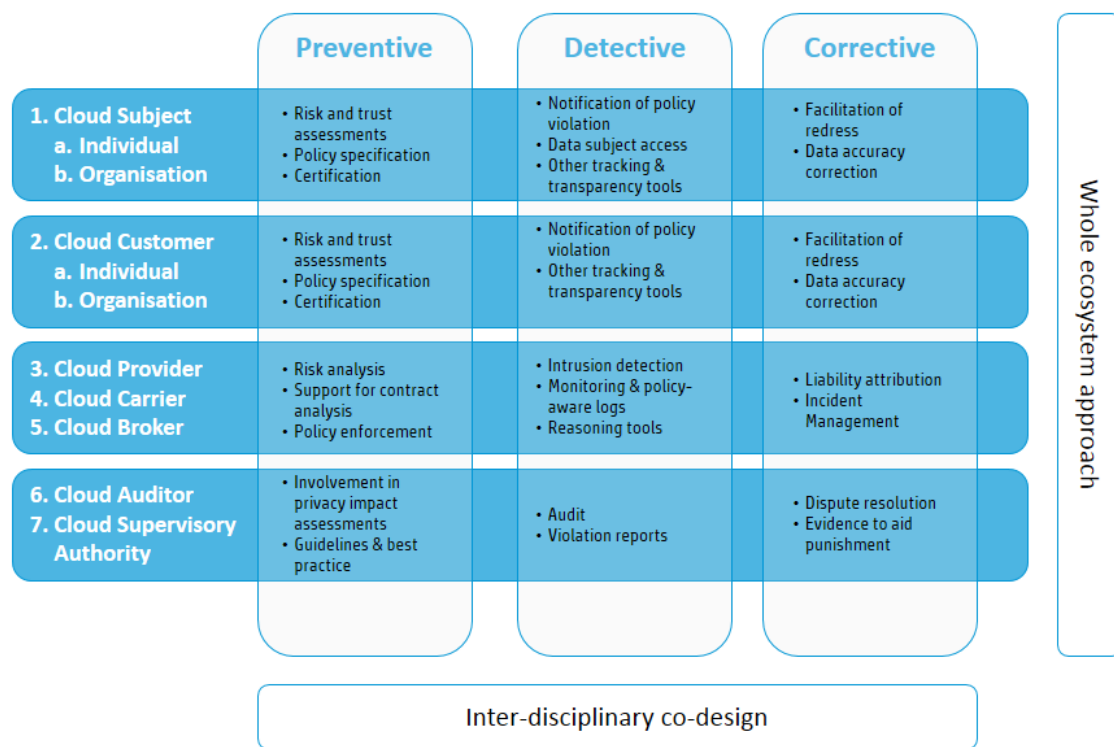


Figure 1 Accountability Framework

This model would vary according to context and depend on relevant parameters. Related questions include investigating: *To what extent do the same mechanisms apply for personal data, confidential data, location info, and other sensitive information? How can we put intelligence into accountability? What should underlie all scenarios? What should vary, in terms of accountability? What guidance can be given in terms of appropriate levels of external assessment and certification for different contextual types?* The functional aspects of accountability may be achieved by mechanisms in the following way:

- **Proper allocation of responsibilities** – via management support, allocation of responsibilities for data protection within an organisation and clarification of responsibilities across supply chains
- **Definition of the contextual obligations to be followed** (carried out by organisations and reflecting stakeholder and regulatory requirements) – via formation of appropriate organisational policies, contracts and stakeholder engagement



- **Risk and trust assessment to decide which mechanisms to use in the given context to meet the policies** – via risk identification/assessment, trust assessment, appropriate choice of business partners
- **Deployment of appropriate privacy and security controls** (these are the mechanisms determined above and include means to make data usages transparent to individuals and to assure that their rights are respected) – via security and privacy best practice, including transparency tools
- **Monitoring data practices** (by organisations and by regulatory oversight) – via tracking tools
- **Detection of policy violations** – via audit and violation detection tools (e.g. evidence collection)
- **Correction of policy violations** – via remediation and/or compensation
- **Reporting of policy violations** – via breach notification and transparency tools
- **Demonstration of policy compliance** (including that policies defined by organisations are appropriate, the mechanisms used are appropriate for the context and that the operational environment is satisfying the policies) – via provision of trustworthy account, verification (about appropriate use of privacy and security controls), certification, provision of evidence about satisfaction of obligations along service provision chains and transparency tools.

Not only are mechanisms to achieve this run within different types of organisation, but others are kinds of meta-mechanisms that can bridge across organisations, for example helping with clarifying responsibilities, or with the verification process. This section reviews some tools and positions them as preventive, detective or corrective mechanisms. Note that other alternative groupings of the mechanisms are possible. For example, from a socio-legal perspective, the accountability mechanisms (in particular, tools) can be grouped according to the type of support (e.g. provisioning of information and advice, controlling compliances and data operations, reporting) given to cloud actors. The main focus would be on detective mechanisms, in particular tools that support operational aspects of accountability in the cloud. Moreover, this section will also position some of the tools currently developed by the Cloud Accountability Project.

### 2.1.1 Preventive Mechanisms

Preventive mechanisms are concerned with supporting accountability before adopting cloud services and during the selection of such services. Among such mechanisms are those that provide information about cloud ecosystems and any potential risks involved. Rather than advising the selection of specific cloud services, such information is intended to support decision-making processes. For example, one of the tools implemented within the Cloud Accountability Project is the Data Protection Impact Assessment Tool (DPIAT), which support cloud customers to perform an analysis of the data protection impact in the case of adopting cloud services, as required by the proposed General Data Protection Regulation (Boyens et al., 2013). Data protection impact assessment would benefit from a risk assessment of cloud ecosystems, or in particular of actors involved in a cloud supply chain. Although conducting a risk assessment across supply chains is envisaged by best practices and guidelines (Boyens et al., 2013), conducting risk assessments of cloud supply chains present various challenges involving different organisations and their roles in cloud ecosystems (Pearson & Yee, 2013).

Other types of mechanisms are concerned with policy specification (and subsequently policy compliance and enforcement). Cloud providers have to comply with different policy obligations derived, for instance, from Service Level Agreements (SLAs) and other legal or contractual commitments. Research and development efforts in this area have mostly focused on translating policy obligations into machine-readable policies. The level of support in terms of tailored accountability mechanisms is yet patchy. Recent developments have extended the same idea to privacy policies. For instance, the Cloud Security Alliance (Cloud Security Alliance (CSA), 2014) is currently developing a Privacy Level Agreement (PLA) that can to a certain extent be linked (translated) to specific machine-readable policies (D'Errico & Pearson, to appear). The ability to comply with relevant data protection policies, that is, to offer cloud services that comply with relevant data protection policies, can be used to identify criteria for analysing alternative contractual cloud offerings. For instance, the Cloud Accountability Project has developed a Cloud Offerings Advisor Tool (COAT), which list cloud services according to security and data protection criteria as specified in cloud offers. This has mainly an informative function of making cloud customers aware of specific security and data protection criteria and supporting their decisions for alternative cloud services. These are just few examples of tools that may support analysis and decision making with regards to emerging threats in the cloud, security and data protection policies and scrutiny of cloud

offers. However, in general, there is limited offers of preventive mechanisms, which also provide a source of evidence in the way organisational practices fulfil the guidelines of frameworks like the NIMITY (Nimity, 2014) and ENISA (ENISA, 2009) ones. In general, most of the support is concerned with the selection of cloud services based on functional requirements and costs rather than security and data protection ones.

### 2.1.2 Detective Mechanisms

Detective mechanisms are concerned with capturing information and highlighting evidence during the operation of cloud services. The majority of tools and services fall within Security Information and Event Management (SIEM) technologies. Gartner analyses industrial offerings in its report on the *"Magic Quadrant for Security Information and Event Management"* (Kavanagh, Nicolett, & Rochford, 2014). The main drivers for such technologies are due to the needs to address security (e.g. threat detections, security breaches) and compliance: *"Broad adoption of SIEM technology is being driven by the need to detect threats and breaches, as well as by compliance needs. Early breach discovery requires effective user activity, data access and application activity monitoring. Vendors are improving threat intelligence and security analytics."* (Kavanagh et al., 2014). Security Information Management (SIM) involves log management, analytics and compliance reporting. Whereas, Security Event Management (SEM) involves real-time monitoring and incident management for security-related events from networks, security devices, systems and applications. SIEM technologies are deployed to support threat management (e.g. real-time monitoring and reporting of user activity, data access and application activity, in combination with effective ad hoc query capabilities), compliance (e.g. log management and compliance reporting) or a combination of such capabilities (Kavanagh et al., 2014).

Among the current SIEM technologies (the leaders in the market as analysed by Gartner (Kavanagh et al., 2014)), are: HP's ArcSight line of SIEM solutions<sup>1</sup>, IBM Security's QRadar SIEM technology<sup>2</sup>, LogRhythm<sup>3</sup>, McAfee (part of Intel Security) Enterprise Security Manager<sup>4</sup>, RSA's Security Analytics, Splunk Enterprise<sup>5</sup>. Gartner highlights that they provide comparable SIEM technologies (Kavanagh et al., 2014):

- HP's ArcSight consists of different SIEM technologies within HP's Enterprise Security Products (ESP) business unit, which also includes HP TippingPoint and HP Fortify. ArcSight Enterprise Security Manager (ESM) software is oriented to large-scale security event management particularly suited to enterprises, ArcSight also offers an appliance-based of ESM that provides preconfigured monitoring and reporting. ArcSight Logger (appliance and/or software) provides log data collection and management functions (implemented stand-alone or combined with ESM).
- IBM Security's QRadar SIEM technology (appliance and/or software) provides log management, event management, reporting and behavioural analysis for networks and applications. It also supports the collection and processing of network flow data and behaviour analysis for different event sources.
- LogRhythm SIEM solutions (appliance and/or software) can be deployed in smaller environments with a single appliance or software instance that provides log management and event management, or it can be scaled as a set of specialized appliances or software instances (log management, event management and centralized console). LogRhythm's Network Monitor supports network forensic capabilities such as data flow monitoring.
- McAfee Enterprise Security Manager (ESM) combines security information management (SIM) and SEM functions, and is available as a stand-alone, all-in-one, virtual appliance and delivered as a managed service by partners. Capabilities can be extended and enhanced with a range of specialized add-on products (e.g. database activity monitoring and analysis, application monitoring, global threat intelligence). McAfee ESM can be integrated with other security solutions tailored to enable context-tailored analysis of vulnerabilities, endpoint state and threats, and to support automated countermeasures.
- RSA (the security division of EMC) Security Analytics (includes enVision and NetWitness) provides log and full packet data capture, security monitoring forensic investigation, and

<sup>1</sup> <http://www8.hp.com/us/en/software-solutions/siem-security-information-event-management/>

<sup>2</sup> <http://www.ibm.com/software/products/en/qradar-siem/>

<sup>3</sup> <https://www.logrhythm.com/>

<sup>4</sup> <http://www.mcafee.com/us/products/enterprise-security-manager.aspx>

<sup>5</sup> [http://www.splunk.com/en\\_us/products/splunk-enterprise.html](http://www.splunk.com/en_us/products/splunk-enterprise.html)

analytics. The Security Analytics reporting system can pull data from both the Security Analytics data structures and the Internet Protocol Database (IPDB) in enVision. The Security Analytics Archiver provides long-term storage and access for compressed logs and log metadata. Security Analytics Warehouse provided big data analytics.

- Splunk Enterprise provides log management, search, alerting, real-time correlation and a query language that supports visualization using statistical commands. Splunk supports log management analytics, monitoring and advanced search and correlation. The Splunk App for Enterprise Security provides predefined reports, dashboards, searches, visualization and real-time monitoring to support security monitoring and compliance reporting use cases. Splunk Cloud (Splunk Enterprise offered as SaaS) allows a cloud provider to search, monitor and analyse machine data that is generated by the infrastructure. Splunk also supports a service for analysing and troubleshooting *cloud* applications.

Other types of detective mechanisms are concerned with cloud service usage (monitoring cloud essential characteristics like resource pooling, rapid elasticity and measured service (Felici & Pearson, 2014)) rather than security and information monitoring. There exists a class of evidence-related cloud technologies that provide generic mechanisms supporting basic (that is, not like the SIEM technologies described) log and monitoring. Examples of such cloud technologies are<sup>6</sup>:

- Sumologic<sup>7</sup> is a log management platform that allows a cloud provider to collect log data from applications (custom, off the shelf), network infrastructure (routers, switches, proxies), and different types of infrastructure (systems, virtualizations and data centres). Log data from firewalls and intrusion detection system. The "cloud powered analytics engine" can be used too. Originally designed for private data centres. Sumologic is offered as a cloud service.
- Amazon Web Services (AWS) CloudTrail<sup>8</sup> is a web service that records AWS API calls for a customer's account and delivers log files to the customer. CloudTrail records important information about each API call, including the name of the API, the identity of the caller, the time of the API call, the request parameters, and the response elements returned by the AWS service. This information helps to track changes made to your AWS resources and to troubleshoot operational issues. The main purpose of CloudTrail is to make it easier to ensure compliance with internal policies and regulatory standards. Amazon CloudWatch<sup>9</sup> offers cloud monitoring services for customers of AWS. The following AWS resources can be monitored in real-time: Amazon EC2 instances, Amazon EBS volumes, Elastic Load Balancers, and Amazon RDS DB instances.
- Cloudlytics<sup>10</sup> is a SaaS product for Analytics & Management of AWS Cloud Logs (currently supporting S3, ELB, CloudFront, CloudTrail and Billing Analytics).
- Logentries<sup>11</sup> is delivered as a SaaS. The Logentries technology collects and analyzes logs across software stacks using a pre-processing layer to filter, correlate, and visualize log data.

Alongside such tools and services, there exist specific development components that are tailored to specific cloud platforms. For instance, the OpenStack Ceilometer<sup>12</sup> collects usage data (utilization of the physical and virtual resources of the deployed cloud services) in order to support billing systems.

Within the Cloud Accountability Project, some of the tools developed are concerned with monitoring data transfer and compliance with data protection policies, monitoring the spread of personal data across multiple services, evidence analyser of compliance with data protection policies. Additionally, the Accountability Audit System (AAS) supports the assessment of audit criteria based on the evidence collected by deployed mechanisms in the cloud.

---

<sup>6</sup> Other examples are: Netiq (audit logs of user activity), Accelops, LogLogic, Sematext, Loggly, PaperTrail.

<sup>7</sup> <http://www.sumologic.com>

<sup>8</sup> <http://aws.amazon.com/cloudtrail/>

<sup>9</sup> <http://aws.amazon.com/cloudwatch/>

<sup>10</sup> <https://www.cloudlytics.com/>

<sup>11</sup> <https://logentries.com/>

<sup>12</sup> <http://docs.openstack.org/developer/ceilometer/>

### 2.1.3 Corrective Mechanisms

Currently, there are no appropriate mechanisms that support corrective actions. From a data protection perspective, it is required to notify customers in case of data breaches (or other threats affecting or that might compromise the security of personal data). However, some technological trends are emerging. These involve different combinations of various technologies, which are individually available in the market but that are starting to get combined together. For instance, there is increasing demand for services/technologies coupling detection mechanisms with corrective (reaction or remediation) mechanisms. One example is the combination of detective security mechanisms (SIEM technologies) combined with software defined networking. This enables organisations to deliver as quickly as possible mitigation actions (e.g. reconfiguration of software defined networks) to emerging security threats. Another example is the combination of SIEM technologies (which usually rely on relative small daily datasets) with historical data (Big Data) and analytics. This enables organisations to develop further understanding of the emergence of security threats.

## 2.2 Frameworks of Evidence Collection

Some recent approaches, propose accountability as a service with the conceptualization of specific evidence frameworks, as it can be found in “Accountability as a Service for the Cloud” (Yao, Chen, Wang, Levy, & Zic, 2010). In general, focus is given to the distributed nature of logs in the cloud, verification or analysis of correctness of service in terms of functioning, and SLA and policies compliance. The authors introduce their concept of accountability, which includes attribution as a major task, and with the core functionality: logging, monitoring and auditing, and dispute resolution. Their approach consists of a novel design to achieve a Trustworthy Service Oriented Architecture (TSOA), to identify and associate failures and misbehaviours with the responsible entities, administrated by a new service, designated Accountability as a Service (AS). The AS service controls the accountability functions, which are kept separate from the operational service domain. Evidence logging is formalised with the definition of evidence event, which is consequently wrapped in its associated meta-information (like timestamp, event description, etc.) forming what is named a trace. Each actor in an interaction keeps local copies of the traces, and the AS keeps a hashed and encrypted version, called token. The logging procedure: logging, authorising, invoicing and execution, is designed to achieve strong accountability. Auditing includes a logic mechanism that verifies compliance based on analysis of pre-established SLAs and business operation logic that defines the correct flow between services. Policy definitions are introduced to fill the gap between SLAs, business logics and monitoring mechanisms used to evaluate service legitimacy, and to define evidence semantics (that is, what should be provided in the tokens, which elements should be used and how they can be used to verify compliance) where services are responsible for doing so. The work includes tests performed on an evaluation system implemented on Amazon EC2.

In the work of (Wang & Zhou, 2010), a collaborative monitoring mechanism is proposed for making multitenant platforms accountable. The proposition considers a third party external service to provide a “supporting evidence collection”, containing evidence for SLAs compliance checking defined distinctively from run-time logs). This type of service is presented as *Accountability services*, offering “a mechanism for clients to authenticate the correctness of the data and the execution of their business logic in a multitenant platform”. The external accountable service contains a Merkle B-tree structure with the hashes of the operation signatures concatenated with the new values of data after occurrence of state changes. The work includes algorithms for logging and request processes, and an evaluation of a testing environment implemented in Amazon EC2.

Digital objects need maintenance due to their dependency on hardware, software and continually changing technology standards. How digital records can be protected, utilized, proved and comprehended over time is conditional on the legal, administrative and technological contexts where they will be taken into consideration. The work in (Blažič, Klobučar, & Jerman, 2007) presents an approach and a solution to tackle the problem related to long-term integrity, authenticity and validity provision of digital data as evidence. The suggested system introduces a standard Trusted Archive System (TAS), based on a digital evidence standard, *Evidence Record Syntax* (Brandner, Pordes, & Gondrom, n.d.). The TAS is established on *PKI-enabled* infrastructures for trust creation, and long-term data integrity proofs, and a service interaction protocol: Long-term Archive Protocol. Some other works of reference on digital evidence are the seminal work on storage formats, Digital Evidence Bags (DEB),



(Turner, 2005) and the extension ontology based approach to correlate event log based evidence (Schatz & Clark, 2006), using semantic web technologies for describing and representing event log based digital evidence.

Butin et al. (Butin, Chicote, & Métayer, 2013), purposes a framework for accountability of practice using privacy-friendly logs. They take the approach of using formal methods to define the “accounts” and the accountability process to analyse these accounts. The sticky policy (i.e. a reference policy) that is attached to the data it refers to and was negotiated between the Data Controller and Data Subject was first formalized which then allows formalizing abstract events as abstract states and semantics, which contains the abstract notions as understood by the data subjects. Secondly, the low-level log events were formalized as concrete states and semantics of concrete events. In the accountability process abstract events and logs are related and the correctness is proved such that an abstract event denotes a concrete log thus allows analysing the abstract event for compliance instead of a log. The compliance with respect to the sticky policy(s) were formally checked and proved. One of the concerns as identified by the authors and a critical concern to realize accountability in a meaningful way is, to verify that the logs which are the basis of any accountability system reflects the actual and complete activities of the Data Controller. In general, their work models an accountability framework that is formally proved and verified which is in contrast to our work in AAS, which is an implementation solving the problem of collecting evidence, analysing them, generating policy-based notifications/violations and generating audit reports.

### 2.3 Scalable Monitoring, Event Processing and Analysis

In the academic literature two directions of work appear in reference to analysing logs for compliance verification. One direction of work concentrates on state of the art methodologies to validate the activities of a system with respect to the policy (either privacy or legal policies) and generates proof of compliance or incompliance. The other direction of work concentrates on analysing the audit trails using intrusion detection techniques to detect known patterns of deviations. Most of the work in this category focuses on the detection of threats imposed by the insiders of an organization. This subsection presents a brief review of the work related to our work in AAS.

Accorsi presents in (Rafael Accorsi, 2008) a concept called privacy evidence to complement the notion of control that is presumably provided by state of the art Identity Management Systems (IMS). Privacy evidence comprises of i) trails of events (i.e., log view) related to the personal data of a data subject and ii) a report, which is a result of an automated audit that compares the log view against the data subject's privacy preferences. In his proposal, the privacy evidences are readily available to the data subjects, and hence provide transparency of the data handling practices of the service providers. Accorsi argues that the concept of a posteriori controls foster confidence of the data subjects on the service providers and promotes the willingness to release personal attributes, which are essential for the personalized services. Accorsi assumes that all the system activities on the personal data are recorded in the form of logs. Counter examples of the privacy preferences are detected by sequentially matching the logs against the privacy preferences of the data subject. He further improves the efficiency of parsing in his subsequent work (R. Accorsi & Stocker, 2008), where a systematic structure called action tree is built from the logs in the pre-processing stage. In the auditing stage the action tree was sequentially pruned against the privacy preferences. The remaining nodes in the tree after pruning are concluded as violations. The results of the parsing (i.e., violations, no violations or uncertainties) are presented using a semaphore notation to improve usability. As identified by Accorsi, one concern in his approach is that it is possible for the data controller to influence the outcome of the audit. Therefore, Accorsi proposes the use of a forward-secure log that makes any tampering with the audit log detectable. In AAS Transparency Log component (Tobias Pulls, Peeters, & Wouters, 2015; Tobias Pulls & Peeters, 2015) provides an evidence store with similar properties.

Similar to Accorsi, Etalle et al. (Etalle & Winsborough, 2007) propose a logical framework to complement the preventive policy enforcement mechanisms. In their work each data object (documents) is accompanied by a sticky policy (usage policy). The users of a system can access, redistribute, receive, and modify the data objects but every action of the users is presumably logged and secured. Later, an audit is performed where the user displays to the auditing authorities (AA) a formal proof that he/she had performed all the actions according to the document's usage policy. If a formal proof cannot be generated for a user's actions because of a divergence in his/her logs, he/she is further investigated for the divergence and held accountable.

Cederquist et al. (Cederquist et al., 2007) in their work take a very similar approach of formal audit procedure for controlling compliance with discretionary access control policies. In their model, there are two components: a proof finder and a proof checker. The user of the system sends parts of the logs corresponding to his/her action that is being audited and the sticky policy to the proof finder to get a formal proof, which is then presented to the auditors to justify his/her action(s). The auditor checks the validity of the proof using the proof checker. However, it is not clear from both Etalle et.al and Cederquist et al. work, to what extent the auditing process is automated, whereas in the AAS the evidences are collected automatically according to the audit tasks. Furthermore, in their work the logs are stored although presumably *secured* in the user's device, which is prone to severe security risks. In the AAS the evidences are stored in the Transparency Log (TL) component, which provides the necessary security properties like forward-secrecy and forward-integrity.

In contrast to the above systems, the objectives of the systems that engage in intrusion detection methods for auditing are to detect:

1. External penetrators – Are those whose intentions are to steal the data or to do some wrongdoing that brings damage to an organization.
2. Internal penetrators – Are those who are authorized users of a system but evade access controls to access sensitive data which is otherwise inaccessible
3. Misfeasors – Are those who are authorized users of a system with higher-level privileges but abuse their privileges.

There are various intrusion detection techniques employed in detecting the threats posed by the above adversaries, but in general the solution techniques are broadly classified into misuse detection and anomaly detection methods. In misuse detection, the audit trails are sequentially checked for known threat patterns, which are to a certain degree similar to the systems described on the top of this subsection, in which the privacy policies are references to detect violations from logs. Nevertheless, attack patterns employed in the misuse detection methods, lacks the granularities of the privacy policies. The anomaly detection techniques on the other hand further are classified into supervised and unsupervised learning techniques. The basic idea is to make the system learn normal patterns from the logs and anything that deviates from the normal pattern is quarantined for further investigation. The methods used in anomaly detection techniques are statistical methods, data mining algorithms like clustering, nearest neighbour discovery, association rules, fuzzy rule etc. and machine learning algorithms using Bayesian model, Markov model, etc. (Patcha & Park, 2007). Because of the maturity of the research in intrusion detection there is substantial amount of literature available to consider, further scientific work like (Lunt, 1988), (Lee & Stolfo, 1998), (Lundin & Jonsson, 2002) summarizes, categorizes, reviews and surveys the researches in intrusion detection.

### 3 Integration of Framework of Evidence by Audit Agent System

In this Section, the relation between the conceptual work on the Framework of Evidence (FoE) and its implementation in the Audit Agent System (AAS) is described. In Figure 2, the conceptual components of the Framework of Evidence are mapped to the AAS architecture. The section marked in **red** deals with automated evidence collection. The **green** section depicts the evidence repository, **blue** is the automated evidence processing and **orange** the presentation of evidence and audit reports. The remainder of the architecture (black) are AAS specific components required for runtime management and agent lifecycle control. A detailed description of the conceptual components is available in (Włodarczyk et al., 2015), whereas a description from an AAS architecture perspective is available in Section 5.

The framework of evidence consists of two main mechanisms: i) Evidence collection, and ii) evidence processing and verification. The evidence collection mechanism focuses on the identification, gathering and storage of evidence elements, used in supporting demonstration of accountability, compliance or violation of obligations, and correctness of practices and controls. The process and verification of evidence mechanisms deal with evidence management, retrieval, verification, and analyses methods, necessary for that demonstration. Details on each of these mechanisms are given in deliverable (Włodarczyk et al., 2015).

Several technical aspects must be considered to support those two mechanisms and the applicability to accountability demonstration. Continuous monitoring of pre-defined sources, transport and security of storage of evidence, analysis and presentation of evidence for audits and internal checks will require specific solutions, able to face the challenges of the distributed nature of cloud infrastructures, virtualization and multi-tenancy problems and concerns. The complexity of all these processes, and also its fast pace, necessarily require a high level of automation and scalability. This can be accomplished by a flexible, specialized but adaptable monitoring system, able to interact with different types of sources and different types of evidences, such as the AAS.

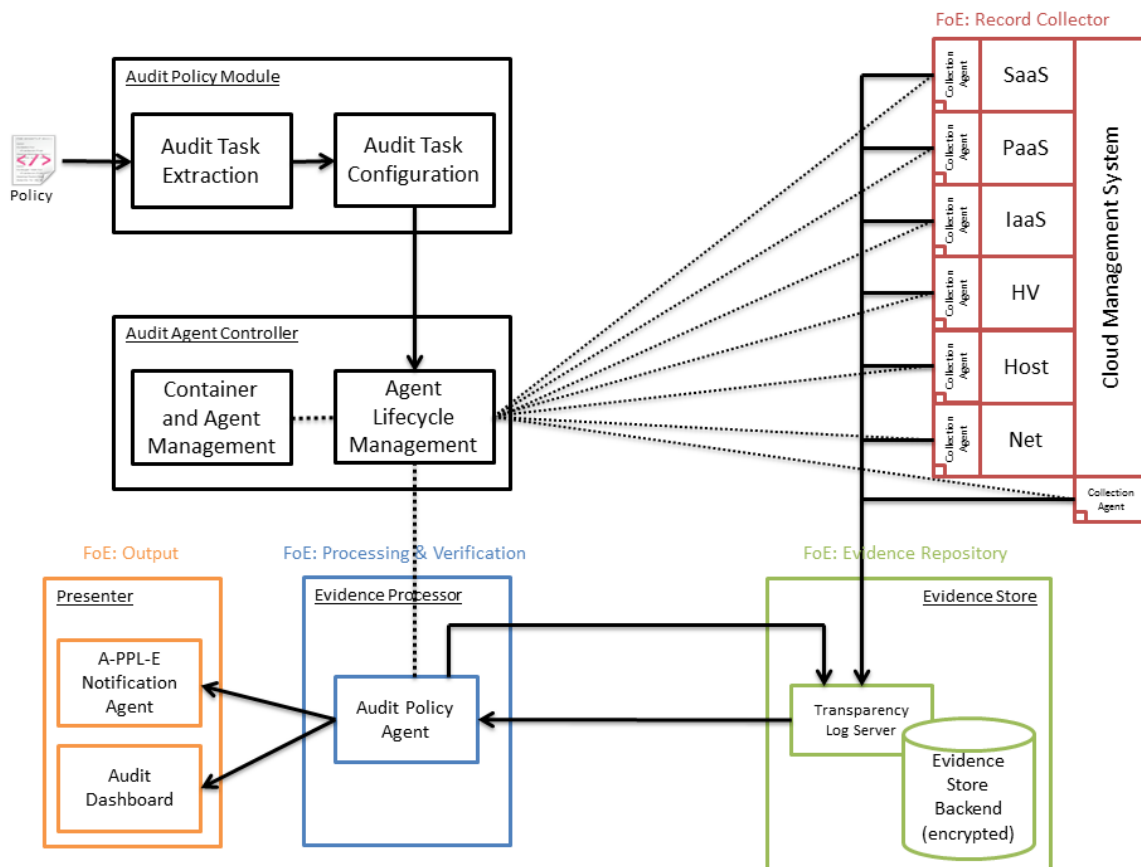


Figure 2 Mapping of Framework of Evidence on Audit Agent System

The most important aspect of evidence collection, in both the FoE and its automation by the AAS is its data sources. The framework considers various evidence sources scattered across a cloud infrastructure. These evidence sources provide elements such as logs generated at different architectural levels and by different tools in the cloud, documentation, cryptographic proofs, configuration settings, files and many more. Because of the highly distributed nature of the evidence sources, the AAS architecture requires software agents to collect that information. These collectors can be seen as probes scattered across the cloud infrastructure collecting evidence.

Since it is in no way feasible to collect data wherever it is generated (this would be problematic from a data protection and a scalability standpoint), only relevant evidence is collected. What constitutes relevant evidence is derived from the input policy, which serves as a basis for describing what obligations are. An input policy can be a security policy, accountability policy, access control policy or any other policy that is machine-readable and from which monitoring and audit tasks can be derived. In AAS the collector agents are configured according to what's described in the policy.

The next important function of the Framework of Evidence, the Evidence Storage, describes several requirements. For instance: protection of the evidence against tampering and unauthorized access. The AAS implements the Evidence Storage as the Evidence Store using Transparency Log as a means for fulfilling the security and privacy protection requirements. Such requirements include for instance the checking integrity of evidence, protection from tampering, protection from unauthorized access etc.

On the Evidence Processing and Verification side of the Framework of Evidence, AAS follows the approach by implementing the capability of generating notifications, generating audit reports (in the form of a web-based dashboard) and the capability to request evidence records on demand. Besides these notification aspects, AAS processing agents provide the capability to include various detection mechanisms such as log analysis, temporal logic, keyword search, event correlation etc. The AAS processing layer is flexible enough to be extended with new analysis tools.



## 4 Scenario

In this Section, we describe how the evidence collection process and AAS are integrated in the demonstration scenario presented in WP-D7. This shows a reasonably realistic deployment of AAS in a multi-provider scenario. Aspects of AAS, such as scalability and provider isolation are described using these scenarios. Additionally, since the processes for evidence collection, processing and audit are generic in nature, a set of obligations, represented in A-PPL with audit information extensions, are used to define demonstration scenarios. These scenarios, based on the obligations, describe evidence collection and evaluation as well as failure states that should be detected in an audit. To cover the most important types of scenarios, we focus on demonstrating key scenarios based on obligations defined in accountability policies (A-PPL) as well as incident detection, which is based on the detection of best practice implementation on the infrastructure level.

### 4.1 Demonstration Deployment

A schematic overview of the AAS deployment in the demonstration scenario, including relevant actors and components is presented in Figure 3. There are three cloud service providers. *DataSpacer* is an IaaS provider. *CardioMon* and *Map-On-Web* are both SaaS providers that don't have their own IT infrastructure but use resources provided by *DataSpacer*. *CardioMon* uses the *Map-On-Web* service to extend its own service. Wearable Co. is a company that produces wearable devices. Customers of Wearable Co. can register with *CardioMon* that provides the corresponding web service to complement Wearable Co.'s service.

In this scenario, each cloud service provider is running its own instance of the Audit Agent System for evidence collection and auditing purposes. The intended user of the system is an auditor. The auditor may act on behalf of the cloud customer (external view) or cloud provider (internal view) to perform continuous, periodic and one-time audits. The goals and nature of policies which are audited may differ depending on the view. The view may also differ in case a trusted third-party auditor (TPA), who is independent from the customer and provider, but acting on behalf of any of those. However, we assume a reasonable degree of technical understanding from the auditor. Therefore, we focus on the internal auditor (e.g., a data protection officer at each provider) and third-party auditor.

At following the AAS is described at each provider.

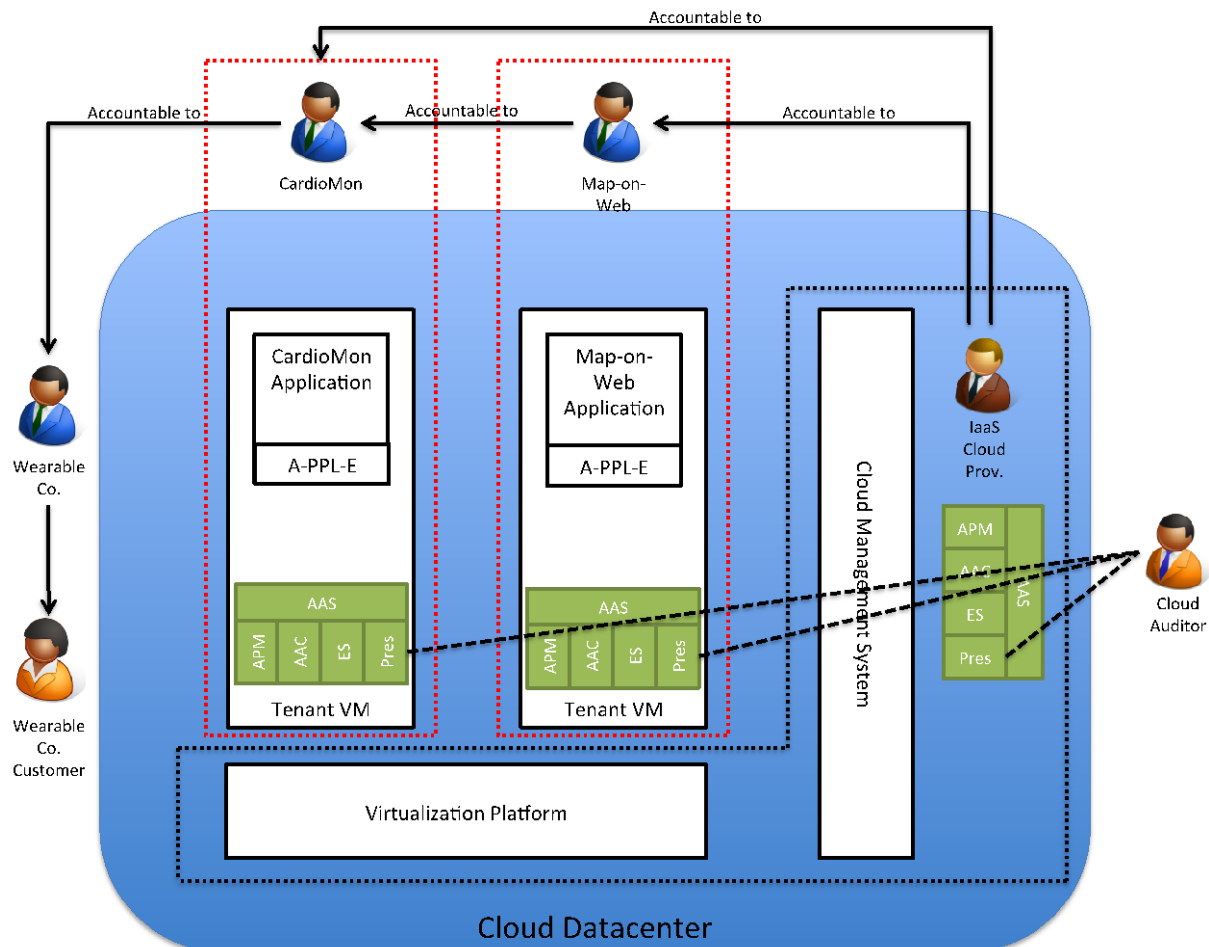


Figure 3 Reference Scenario Overview (Demonstrator)

#### 4.1.1 CardioMon Audit Agent System (AAS)

The SaaS provider *CardioMon* runs AAS in addition to its own service on the virtualized resources provided by *DataSpacer*. This means, inside its own administrative domain (i.e., where *CardioMon* has full control of the resources), *CardioMon* collects evidence of its operations. In this case, this mainly concerns evidence collection and audit inside *CardioMon*'s virtual machines. The main user groups of *CardioMon*'s AAS instance are: *CardioMon* customers, internal auditors and external third-party auditors. However, *Wearable Co.* does not have the technical expertise to conduct audits on *CardioMon* itself, therefore internal and external third-party auditors are more important.

##### Internal use of AAS by *CardioMon*

- General auditing of the *CardioMon* service operation (e.g., availability, access control)
- Auditing of policies agreed upon between *CardioMon* and its customers (self-audit)

##### External use of AAS by *Wearable Co.* / third-party

- Auditing of policies agreed upon between *CardioMon* and *Wearable Co.* The naïve customer is not doing audits, but relies on third-party audits by cloud auditors and DPAs.

#### 4.1.2 Map-on-Web Audit Agent System (AAS)

The SaaS provider *Map-On-Web* runs AAS in addition to its own service on the virtualized resources provided by *DataSpacer*. This means, inside its own administrative domain (i.e., where *Map-On-Web* has full control over resources), *Map-On-Web* collects evidence of its operations. In this case, this mainly concerns evidence collection and audit inside *Map-On-Web*'s virtual machines. The main user groups of *Map-On-Web*'s AAS instance are: *Map-On-Web* customers (most importantly *CardioMon*), internal

auditors and external third-party auditors. Although *CardioMon* does have the technical expertise to conduct audits on *Map-On-Web* itself, the focus is on internal and external third-party auditors.

#### Internal use of AAS by *Map-on-Web*

- General auditing of the *Map-On-Web* service operation (e.g., availability, access control)
- Auditing of policies agreed upon between *Map-On-Web* and its customers (self-audit)

#### External use of AAS by *CardioMon*

- Auditing of policies agreed upon between *Map-On-Web* and *CardioMon*

### 4.1.3 *DataSpacer* Audit Agent System (AAS)

This infrastructure provider provides both *CardioMon* and *Map-on-Web* with resources required to run their services (virtual machines, storage and networking). Since *DataSpacer* acts as an independent cloud service provider, it also runs its own Audit Agent System installation to monitor its operation, collect evidence according to its operations policies and policies agreed upon with its customers. There are two major user groups, which act as auditors and interact with *DataSpacer's* AAS Dashboard. For one, *DataSpacer* personnel (e.g., administrators) act as internal auditors. They focus on monitoring and auditing using AAS. The second user group are customers of *DataSpacer* (e.g., *CardioMon* and *Map-on-Web*). They use *DataSpacer's* AAS to monitor and audit *DataSpacer* with a strong focus on their agreed upon policies and resources used by the customer.

#### Internal use of AAS by *DataSpacer*:

- Availability monitoring on infrastructure level
- General auditing of cloud management system operations (e.g., resource management, access control)
- Intrusion detection

#### External use of AAS by *CardioMon*, *Map-on-Web* and third-party

- Availability monitoring on virtual machine and storage volume level
- Auditing of cloud management operations on per-tenant basis
- Auditing of policies agreed upon between *DataSpacer* and *CardioMon* or *Map-On-Web* respectively

## 4.2 Scenario A: A-PPL-based

The A-PPL-based Scenarios focus heavily on data handling obligations. This includes access control, usage control and general data handling. In these scenarios, an A-PPL policy is used as input for:

- Evidence Source: the location and entity producing evidential data. Since this information is typically not available explicitly in A-PPL, manual complementation by the auditor is required.
- Potential Checks: the type and number of checks that need to be performed in order to check for compliance or violation of an obligation.
- Thresholds in Audits: detailed information on what may be considered a violation for the aforementioned checks.

In the following, several evidence collection and audit scenarios are presented.

### 4.2.1 Scenario A1: Data Retention

#### Description

*Data retention obligations* play an important role in data handling. The A-PPL Engine enforces retention obligations based on individual datasets by tracking relevant timestamps and deleting datasets if the maximum retention time has been reached. Therefore, the A-PPL Engine is at the central of data handling in the cloud service. Here, it enforces policies such as retention obligations and has full control over the datastore containing the personal identifiable information (PII) datasets in question. However,

there are cases, where datasets can be cloned without A-PPL Engine being aware of it as following. In these cases, the PII dataset effectively exists outside the control of A-PPL. Such cases include:

- Data duplication on a lower-level cloud layer (e.g. snapshot using the Cloud Management System, Backups)
- A-PPL access control circumvention (e.g. files stored in a backup store)

While A-PPL Engine effectively controls and enforces policies on the PaaS and SaaS layer, operations on the IaaS layer are out of scope of A-PPL Engine. This means that low-level operations, such as virtual machine and virtual storage snapshotting or cloning operations respectively, may circumvent A-PPL-E's data retention enforcement. Snapshotting preserves the state of an object (e.g., VM or block/object storage) at a specific point in time. Snapshots are, by definition, immutable. Therefore, any delete operation issued in the original object (e.g., a dataset is deleted) is not reflected in the snapshot.

#### **Detection Mechanisms**

- Data duplication audit by AAS
  - Audit of CMS-recorded operations that may duplicate data (snapshot, clone)
  - Audit of complementary services, such as backup services and operations
  - Breach detection might include audit of VM accesses / privileges, but may not be detectable on this low level

#### **Communicated Information**

- Reference to possibly violating event from CMS
- Data retention failure

#### **4.2.2 Scenario A2: Notification**

##### **Description**

*Notification obligations* play an important role in data handling. The A-PPL Engine enforces notification of stakeholders in case of detected policy violations and data breaches. A-PPL-E is capable of notifying stakeholders in different ways (e.g., by email or Transparency Log). Additionally, the notification obligation may be linked to a time constraint (e.g., a notification has to occur in the first hour after detection).

While A-PPL-E can send notifications, it is out of its scope to make sure notifications were actually sent and sent in time.

##### **Detection Mechanisms**

- Notification mechanism monitoring and audit
  - Audit of operations recorded by the notification tool (e.g., mail transport agent (MTA) log)
  - Timestamp analysis and comparison with corresponding policy violation

##### **Communicated Information**

- Detail of failed operation
- Notification failure

#### **4.2.3 Scenario A3: Right to Know vs. Need to Know**

##### **Description**

Access rights on personal data are enforced in the A-PPL-Engine. However, there are cases, where an individual access request is granted (necessary privileges available) but context information (e.g., large number of access requests in a short period of time) indicates a violation (e.g., malicious insider, hacked account). Monitoring such scenarios is typical for intrusion detection systems (IDS). AAS agents can be used to interface with existing intrusion detection systems (e.g., register themselves as receiver of alarm events, polling for events) or implement intrusion detection mechanisms/tools themselves (e.g., log analysis component).

##### **Detection Mechanisms**

- Intrusion detection (on database entries, API calls, network traffic monitoring)

- Behaviour analysis (learning data access patterns)
  - Rule-based, what kind of thresholds of events (no. of operations in a given time frame) triggers flagging/investigation/blocking
- Log analysis (specifically error logs)

#### **Communicated Information**

- Security-aware recipient (CSP administrator)
  - Timestamps
  - Details of intrusion (who, what, reason for alarm etc.)

### **4.3 Scenario B: Incidents**

Incident scenarios focus on detection of security, privacy and availability incidents in the cloud infrastructure, which can not necessarily be linked to an A-PPL policy. In this case, templates for incident detection are pre-defined in AAS and provided out of the box. However, since there is no policy input, most of the configuration has to be performed manually by the auditor (e.g., thresholds, IP configuration etc.). In the following, three scenarios are described that demonstrate this class of evidence collection and audits.

#### **4.3.1 Scenario B1:SSL Scare**

##### **Description**

Misconfiguration of services and failing to patch software quickly can lead to severe security problems such as being vulnerable to exploits and violating security requirements. Recent SSL vulnerabilities such as POODLE (CVE-2014-3566, 2014), BEAST (CVE-2011-3389, 2011) and Heartbleed (CVE-2014-0160, 2014) are prime examples for the need to patch as soon as fixes become available. However, patching may not be enough in some cases. For instance, to mitigate the Heartbleed vulnerability, certificates need to be replaced, old certificates revoked and private keys changed. Besides that, problems can arise from service misconfiguration. The recently discovered POODLE vulnerability is closely linked to obsolete protocols being allowed (which is an SSL configuration problem). Also, in cases where strong cryptography is required, specific SSL configuration is required (protocol versions, available cipher suite, cipher order, algorithms, key length, certificate status etc.). AAS can be used to perform checks on both patch management (checking a server's installed packages and comparing versions against a known-good list) and configurations (checking previously described parameters in service configuration files). Additionally, in cases where files cannot be accessed directly, agents can perform vulnerability checks externally by using port scans, protocol validation and performing security scans/vulnerability checks.

##### **Detection Mechanisms**

- Internal service audit:
  - File alteration monitoring and triggered configuration file audit (e.g., administrator error)
  - Keyword search, parameter checks in configuration files
  - New/changed files (e.g., certification for file changes)
  - Software version checks (checking critical patches ,using the packet management system)
- External service audit:
  - Port scan (only secure configuration visible externally, e.g. no plain text connection possible)
  - Protocol validation (valid certificates, valid/secure cypher suite, recommended key properties (size etc.), secure protocol versions)
  - Vulnerability check: e.g., specific agent for detecting POODLE, BEAST, Heartbleed etc.

##### **Communicated Information**

- Security-aware recipient (CSP administrator, customer administrator)
  - Timestamp
  - Vulnerability assessment with respect to known attacks
  - Details of configuration parameters in violation of security policy (config lines; software version mismatch; not who made the changes since this may not be known)

#### 4.3.2 Scenario B2: Service Availability

##### **Description**

Cloud computing can be used to improve availability (by buying redundancy, qualified administrators, distributing across data centres etc.). However, it is important that availability is clearly defined at the beginning (considering maintenance time, etc.) However, there can be service outages and hardware failures are quite common due to focus on consumer-grade hardware. In some cases, hardware failures can lead to lack of service availability. This is a common problem and is addressed by using redundant systems. When redundancy fails (e.g., whole data centre unavailable and no off-site) user's need to be informed accordingly. AAS can be used to gather availability information, performance counters etc. from various sources, such as the cloud management system (e.g., OpenStack), a dedicated server monitoring tool (e.g., Nagios) and from custom probes (e.g., simple ICMP probe, application level probe) to detect service availability incidents.

##### **Detection Mechanisms**

- Infrastructure/hardware monitoring (server, network components etc.)
- ICMP probes/agents at network level (very low-level, initial indicator, but not adequate)
- Higher protocols probes/agents at application level (e.g., HTTP, automated service login)
- Hardware failure that does not necessarily impact service availability immediately (redundancy)

##### **Communicated Information**

- Timestamp
- Service availability incident (application-level check result)
- "Component" availability incident details (only if it impacts service availability)

## 5 Evidence Collection Service

In this Section, we present the evidence collection and audit service, the Audit Agent System (AAS). The Audit Agent System implements the Framework of Evidence, its components, mechanisms and processes in an efficient and scalable way. Before the subsections “Proof of Retrievability”, “Automated Incident Detection”, and “Automated Evidence Collection” are described, the “Architecture of the Audit Agent System (AAS)” is outlined.

### 5.1 Architecture of the Audit Agent System (AAS)

The AAS architecture comprises six major modules that are described in detail in the following subsections. The six modules can roughly be divided into four major parts:

1. *Input*: Audit Policy Module (APM)
2. *Runtime Management*: Audit Agent Controller (ACC)
3. *Collection and Storage*: Evidence Collection Agents, Evidence Store
4. *Processing and Presentation*: Evidence Processor, Presenter

#### 5.1.1 *Input*: Audit Policy Module (APM)

The Audit Policy Module (APM) is the main component for handling input to the AAS. Typically, obligations, access control requirements and other types of policies define how a cloud service is supposed to handle data. To gather evidence about the compliance with or violation of these policies is part of the AAS. In the APM, machine-readable input policies are parsed and evidence collection tasks as well as evidence processing tasks are extracted (see also Section 6 for a detailed description about the extraction of audit tasks). The main assumption in this parsing process is that this will not be fully automatable. Therefore, additional information is provided by the auditor. Depending on the actual audit task, this includes infrastructure-specific information such as:

- Specifics of the evidence source (IPs, JADE agent platform, REST endpoints)
- Specifics of the monitored service (path to log files, files to monitor for changes)
- Required credentials (authentication strings, usernames, passwords)
- Audit type (periodic, continuous, one-time)

#### 5.1.2 *Runtime Management*: Audit Agent Controller (AAC)

The Audit Agent Controller (AAC) is the main runtime management component. At the core of AAS it is responsible for orchestrating audits and agents according to what has been previously defined in the APM. The typical audit lifecycle is as follows:

1. According to the input provided by the APM, AAC creates and configures audit policies, its tasks and corresponding collection and processing agents.
2. Agents are migrated between the core platform and target platforms (near the evidence source)
3. During the agents' lifetime, the AAC monitors registered platforms and registered agents, handles exceptions, manages the creation, archival and deletion of evidence stores

The AAC uses the JADE Agent Communication Language (ACL) (JADE, 2014) for internal communication between agents. Therefore, the AAC sits at the core of the AAS and manages all operations regarding the orchestration of collection and processing agents, as well as maintaining the Evidence Store. Most notably, the AAC uses UDP-based monitoring of the various agents to ensure a consistent and smooth operation of the AAS.

#### 5.1.3 *Collection and Storage*: Evidence Collection Agents, Evidence Store

From the various sources of evidence in the cloud, evidence records are collected that will be stored in the Evidence Store on a per-tenant (i.e., per cloud customer) basis located on a remote server. The



evidence sources are considered to be logs, cryptographic proofs, documentation, certificates, and many more. The AAS architecture is built around using software agents for evidence collection, evidence evaluation and controlling the overall system. Agent technology helps ensuring extensibility by allowing easy introduction of new evidence sources by adding new collection agents. This approach also allows AAS to address rapid infrastructure changes, which are very common in cloud infrastructures by easily deploying and destroying agents when needed.

There are various evidence sources to be considered, such as logs, cryptographical proofs, documentation and many more. For details on the various kinds of evidence sources, refer to (Włodarczyk et al., 2014). For each, there needs to be a suitable collection mechanism, for instance, a log parser for logs, a tool for cryptographical proofs or a file retriever for documentation. This is done by a software agent called Evidence Collection Agent, which is specifically developed for the data collection from the corresponding evidence source.

The Evidence Collection Agent reads raw evidential data from the source and generates evidence records that are sent to the Evidence Store. The Evidence Store is implemented using Transparency Log (TL) (T. Pulls & Martucci, 2014) (Ruebsamen, Pulls, & Reich, to be published). Since TL functions as a key-value store for storing evidence records (encrypted messages identified by a key) NoSQL or RDBMS-based backends for persisting evidence records can be used. All data contained in the Evidence Store is encrypted. The evidence records are encrypted on a per audit task basis, which means only the Audit Policy Agent corresponding to the Collection Agents is able to decrypt the evidence records for further processing. Isolation between tenants in a single Evidence Store is achieved by providing one container for each tenant where his evidence records are stored. However, even stronger isolation by providing a separate Evidence Store hosted on a separate VM is also possible with this approach.

#### 5.1.4 *Processing and Presentation: Evidence Processor, Presenter*

The processing or analysis of evidence consists of two steps:

1. Retrieve the appropriated collected information from Evidence store (which must be policy/audit task based).
2. A verification process, which checks the correctness of recorded events according to defined obligations and authorizations.

These procedures are inherently dependent on the type of audit task. There can be specific audit tasks defining a single or a small set of checks to be performed (e.g. availability of VMs, results of a PoR, etc.), or more complex compliance over time periods (e.g. monthly checks of policy compliance). According to the complexity of task, due to amount of obligations, or the volume of evidence to analyse, different verification processes may need to be considered, ranging from log mining, checking for predefined tokens or patterns, to automated analysers and automated reasoning upon the audit trail.

For the situations when the audit task consists of defined checks, the evidence store is accessed and the required logs (or other elements) are identified in the related evidence records. The more complex compliance check will involve the retrieval of evidence records covering given periods of time, or specifically related to a policy ID.

The scenarios included in this deliverable (see Section 4.1), and the automated evaluator presented in (Włodarczyk et al., 2015), illustrate both types of verification processes where general auditing of service operation, such as availability, access control, resource management, etc., and internal and external auditing of agreed policies, are considered.

The outputs of any audit, including report, notification alerts, and messages of non-compliance, are then processed for presentation.

There are two main ways of evidence presentation in AAS. The A-PPL-E Notification Agent is designed to generate violation notification messages, which are consumed by the A-PPL Engine. The A-PPL-E Notification Agent uses A-PPL-E's REST interface to report policy violations. From there, A-PPL-E can proceed with the reported violation according to what's defined in the A-PPL policy.



The second presenter in AAS is the web-based dashboard, later shown in Section 6.5 “Audit Task Example”. The Auditor uses the Dashboard as the main way of interaction with AAS. Most importantly, audit results are displayed to the auditor, which provides an immediate overview of the current compliance status. The main contact point with the system is the audit dashboard which is a web application implemented using Bootstrap in cooperation with WP-D5 to achieve a uniform look-and-feel of A4Cloud tools and an optimal user experience.

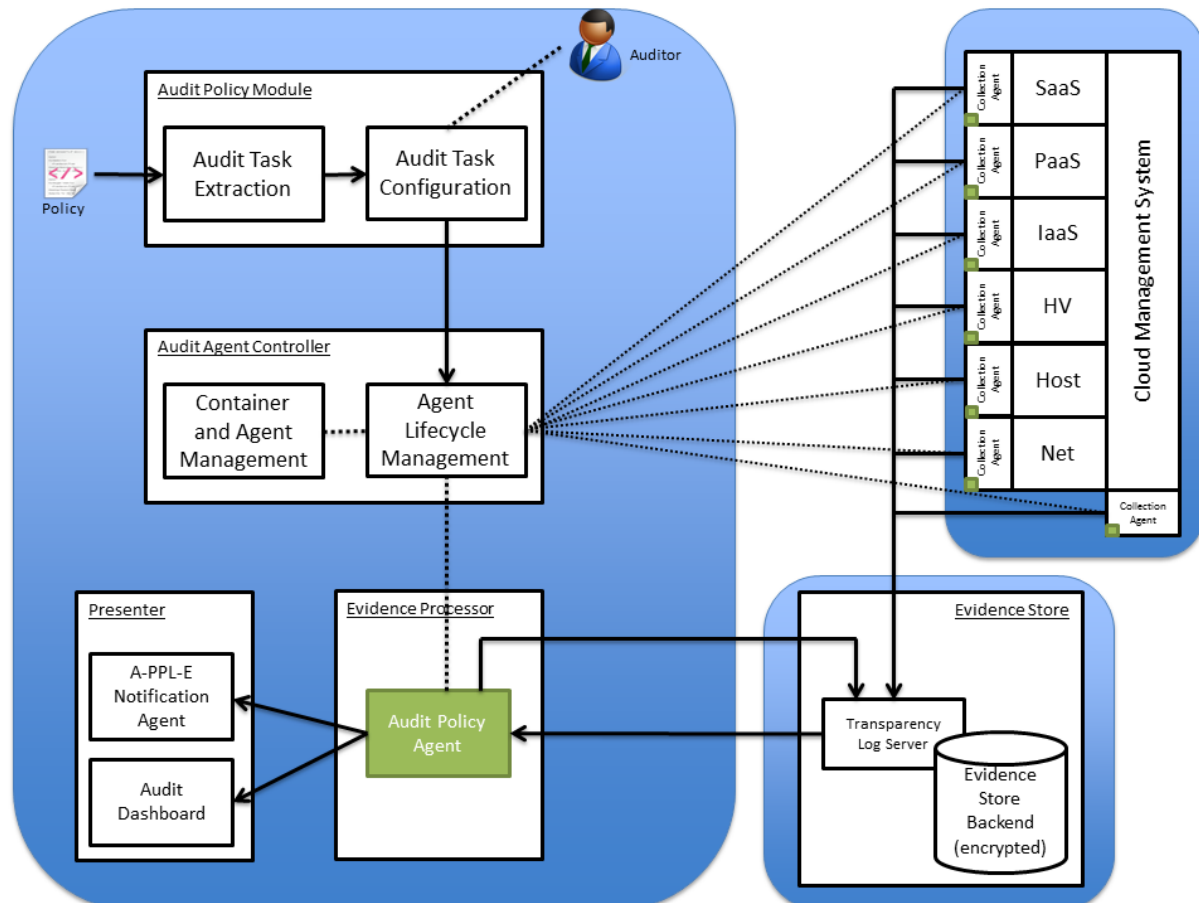


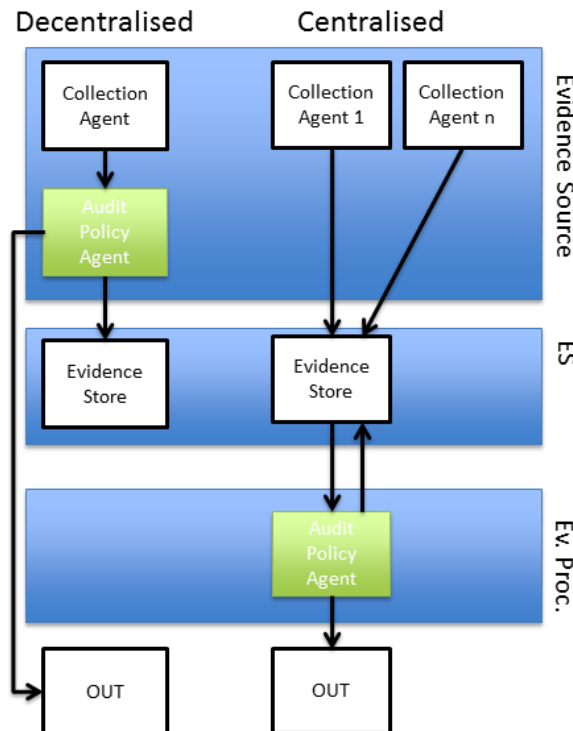
Figure 4 AAS Architecture Overview

#### Automated Incident Detection

Collected evidence serves as the basis for policy violation detection in AAS. Audit agents monitor collected Evidence Records and generate violation or compliance notifications. There are three modes an audit agent can run in:

Continuous: in continuous mode, the audit agent evaluates evidence records as soon as they are generated by the collection agent. The continuous audit mode is very similar to monitoring with immediate notification if a violation is detected. The time between evidence about a violation or incident being recorded and actual detection and notification is minimal in this scenario. However, since evidence is analysed on-the-fly, more complex evidence analysis that relies on taking a whole series of records into account is generally harder to implement.

- Periodic: in periodic mode, the audit agent evaluates evidence records at specific intervals (e.g. hourly, daily, weekly etc.).
- One-time: in one-time mode, the audit agents, collection agents and the corresponding evidence records are archived immediately after the audit result has been generated.



**Figure 5 Approaches for Automated Policy Violation Detection in AAS**

In Figure 5, we describe the processes for automated policy violation detection in AAS. In general, the phases of automated policy violation detection are similar to the phases of a typical audit and include:

1. Evidence collection: in this phase, dedicated collection agent gather information from various sources.
2. Evidence storage: in this phase, evidence records are stored for evaluation at a later point in time.
3. Evidence evaluation (audit): In the evaluation phase, evaluation agents analyse evidence records to perform incident and policy violation detection.
4. Reporting: in the reporting phase, audit reports and policy violation detection notifications are generated and emitted.

Since all functional aspects of AAS are implemented as software agents, this allows for much more flexibility than in a monolithic architecture. In this particular case, even the AuditPolicyAgent, which is the main agent reading evidence records from the Evidence Store and evaluating those records against what is defined in the input policy, can either run on the AAS core or can be distributed to be placed near the actual evidence producing entity in the cloud (i.e., the same runtime environment than the evidence collection agent). These two scenarios are depicted in Figure 5. In the centralised approach, multiple evidence collection agents generate evidence records and store them in an Evidence Store. From the evidence store, an AuditPolicyAgent reads a set of records in a regular interval and performs its analysis. Upon detection of a violation, a notification and audit report entry are produced. In the aforementioned scenario B1 this means, there are multiple collection agents deployed: an agent for monitoring A-PPL-E PII requests and whether or not they are encrypted and an agent for monitoring the service interface configuration for secure parameters regarding available cryptographic algorithms, key material and validity of cryptographic certificates. This monitoring and audit information is saved in the Evidence Store, from which the AuditPolicyAgent reads during the evaluation phase.

In case of simple audit policies, an AuditPolicyAgent can also run decentralised and interact with the collection agent directly. This scenario is also depicted in Figure 5 (left). Essentially, the flow of data is similar to the centralised scenario, but requires less communication between agents, less transfers of data over the network and less computing resources on the central platform. By allowing the collection and audit agents to interact directly, the evidence store is reduced to an archive of evidence and audit results that can be used later on to gather more details about what has happened.

## 5.2 Proof of Retrievability (POR)

The framework of evidence described in (Włodarczyk et al., 2014, 2015) includes the provision of a particular source of evidence that relies on cryptographic techniques. Whereas audit logs provide evidence for a sequence of monitored events and actions in an accountable system, cryptographic proofs offer another type of evidence proving at a given moment the assurance of the correct behavior of cloud actors. These proofs are requested, collected and analysed by AAS in an automated way, giving opportunity to check that the cloud complies, at any point of time, with accountability obligations. The authors in (Włodarczyk et al., 2014, 2015) develop the need for such cryptographic proofs in the proposed framework of evidence. In this section, we present a practical case of the automation of evidence collection for the data at rest scenario.

In particular, proofs of retrievability (PoR) are an example of such cryptographic proofs that have practical relevance for the cloud environment and that can be suitable for any cloud services offering storage capabilities. Moreover, their collection and verification can be automated and integrated in an auditing tool such as AAS. In a nutshell, PoR gives assurance of compliance (or non-compliance) of the obligation related to the storage of data by CSPs. Therefore, these proofs focus on the data at rest scenario, where the owner of the outsourced data is concerned by its retrievability, a notion summarized as the combination of integrity and availability. Hence, PoR are for cloud customers a vital obligation, since storage services are one of the most widely offered services in cloud computing. Indeed, to cope with data loss incidents, reported as the second biggest cloud computing threat in the CSA's Notorious Nine (Cloud Security Alliance, 2013) and which may occur as a result of system failures, catastrophic disasters or malicious attacks, any cloud user may wish to verify, at any point of time, the retrievability of its outsourced data, while still enjoying cloud computing benefits. For the sake of a scalable solution to the problem of data retrievability in the cloud, we advocate for the provision of cryptographic Proofs of Retrievability, as a potential source of evidence for correct data storage. PoR have the benefits to handle the peculiar challenges of cloud computing: PoR enable the verification of big data without the requirements of (i) physical possession of the outsourced data, (ii) heavy computations at the user side and (iii) substantial storage overhead at both the user and the cloud sides. In the context of the A4Cloud project, we propose StealthGuard (Azraoui, Elkhayaoui, Molva, & Önen, 2014), a new solution for the provision of PoR. After giving a general and theoretical description of StealthGuard (see Section 5.2.1), we show how our PoR protocol integrates in two A4Cloud tools, namely in the A-PPL-Engine (see Section 5.2.2) and AAS (see Section 5.2.3).

### 5.2.1 StealthGuard

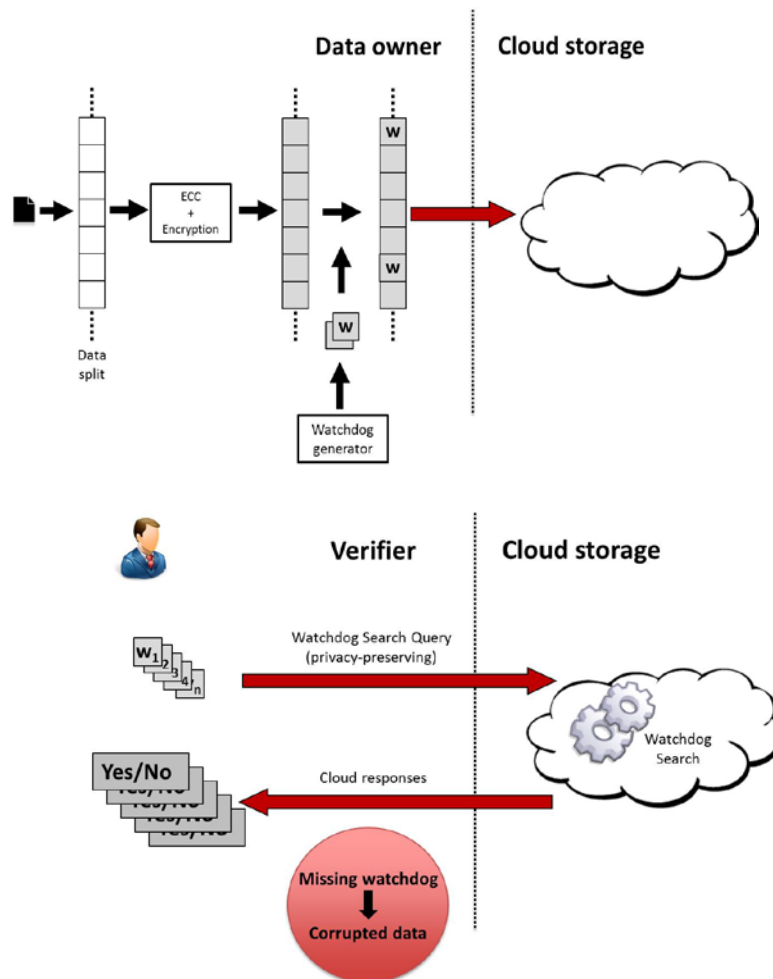
StealthGuard (Azraoui et al., 2014) is proposed as an efficient PoR protocol that enables any cloud user to check that the data he has outsourced is correctly stored by the cloud service provider, that he can gather some proofs of its retrievability at any point of time.

StealthGuard is divided in three phases described as follows:

- **Setup Phase:** To outsource a retrievable version of its data, the owner performs a set of operations enabling retrievability checks. In particular, the data owner generates a certain number of random blocks called watchdogs that are inserted in random positions in data. Thanks to the pre-processing of the data, the watchdogs are indistinguishable from the original data and hence, no one can detect their existence. The resulting file is then sent to the cloud. The owner can delete it from its local storage and keeps only the keying material that is used during this setup phase.
- **Audit Phase:** At a certain point of time, a verifier wants to check the retrievability of the outsourced data. This verification consists in searching for some watchdogs inserted in the previous phase and in deciding whether they are still intact in the data. This phase leverages a watchdog search solution based on a privacy preserving word search protocol. The search enables the verifier to send lookup queries without letting the storage server derive any query information from the query nor from its result. The search results are sent back to the verifier.
- **Verification Phase:** Once the verifier receives them, he verifies the cloud responses and makes a decision about the presence of the watchdog in the outsourced data. If the results are correct, then they prove, with a certain probability, that the targeted watchdog is intact in the data.

Otherwise, they prove that it has been corrupted. The audit-verification phase is repeated a certain number of times on different watchdogs in order for the verifier to decide

**Figure 6 PoR setup (above) and PoR audit (below)**



whether the file is retrievable. More details about StealthGuard can be found in (Azraoui et al., 2014). We integrate the above PoR protocol in AAS and A-PPLE. In addition, as an essential source of evidence in the FoE, the provision of proofs of retrievability will be monitored by creating a set of records that will give account of the issued PoR queries and responses.

### 5.2.2 Integration of Proofs of Retrievability in the A-PPL Engine

As part of the A-PPL language, we create two new language elements `TriggerPORRequestReceived` and `ActionPOR` that govern the enforcement of a PoR collection defined in an accountability policy written in A-PPL. The idea is to enable the verifier to send a PoR request to the engine which will launch `TriggerPORRequestReceived`. This trigger in turn fires the action `ActionPOR` that rules the generation of the requested PoR. Once the PoR are generated, they are sent back to the verifier who checks their correctness and makes the decision on the retrievability of the data.

For this purpose we include in the engine the PoR operations related to the enforcement of accountability policies that may include PoR rules. The enforcement of A-PPL policies are handled via the A-PPL engine. In a nutshell, the engine has been implemented to parse A-PPL policies and enforce access and usage control rules and accountability policies related to notifications, logs, encryption of data and audits. For details concerning the engine please refer to the deliverable (Santa de Oliveira et al., 2015). For the sake of description of the extensions we add to the A-PPL engine, we follow the three phases of StealthGuard, our proposed PoR protocol, described in the previous section.

**Setup phase:** As mentioned before, StealthGuard first requires a pre-processing step that yields a retrievable version of the data to be outsourced. In this regard, we extend the client module interfacing with the A-PPL engine with the function called `SetupPOR` which takes as input the “plain” data and outputs the retrievable version of that data. This function `SetupPOR` therefore applies the ECC and the encryption, and inserts the watchdogs. Once these operations have been performed, the module uploads the modified data to the engine.

**Audit phase:** The operations performed by a verifier will lie in the A-PPL engine client module. In this phase, a single operation consists in the generation of a set of PoR queries targeting a particular piece of data and in the sending of these queries to the engine. Therefore, we extend the module with the function `PORVerification` which handles the query generation. Once these queries are well generated, they are sent to the engine which activates `TriggerPORRequestReceived`. At this step, the `ActionPOR` action is fired. It consists in two operations: (i) the generation of the requested PoR, which, we recall, are the outputs of the private search for the requested watchdogs, as mentioned in section 5.2.1. and (ii) the sending of the PoR responses to the client module. Therefore, we extend the engine with the function `PORAction` that implements the operations ruled by the policy action `ActionPOR`.

**Verification phase:** The whole process of this phase is handled in the engine’s client module through the function `PORVerification`. This function checks whether the received PoR are correct by comparing the received values with the expected values that the verifier can regenerate.

### 5.2.3 Integration in AAS

The advantage of the abovementioned integration of StealthGuard in the A-PPL engine lies in the fact that all the client’s operations are located in a dedicated and standalone module, namely the engine’s client module. Therefore, we envision porting this module inside the AAS for the verification of proofs of retrievability.

As depicted in Figure 7, the A-PPL engine’s module where all the PoR verifier’s operations are performed can be used by the AAS to collect proofs of retrievability from the A-PPL engine (which is the PoR prover).

Additionally, when an auditor wants to check the retrievability of a particular piece of data, he will issue a PoR query. A record will be created by the record collector with the attributes described in (Włodarczyk et al., 2014). Similarly, when the prover generates the requested proofs and sends them back to the verifier, a record will be issued and stored in the record database, giving account of the PoR procedure. (Włodarczyk et al., 2014) lists the attributes that are recorded for such an event.

To perform a Proofs of Retrievability protocol via AAS, we follow the steps of StealthGuard mentioned above. Each of the new functionalities added in the engine are called by the AAS that runs the client’s module.

**Setup phase:** On behalf of the data owner, AAS processes the data by calling the function `SetupPOR`. Depending on scenarios, this phase can be performed independently from the AAS. However, to be able AAS to perform audits and verifications, the AAS requires the keying material generated during this phase. Therefore, we advise the Setup phase to be done via the AAS and the client module of the engine.

**Audit phase and verification phase:** AAS collects the so-called proofs of retrievability by calling the functionalities implemented in the standalone client module of the A-PPL engine. Namely, to produce a POR request and to verify the corresponding POR response sent by the cloud, AAS calls the function `PORVerification`. Note that AAS should communicate with the A-PPL engine of the audited cloud storage. This is done through a RESTful HTTP connection between the two entry points in the engine (specifically the client module and the server’s entry point). Additionally, the results of the POR verification can be displayed to the user through AAS’ user interface.

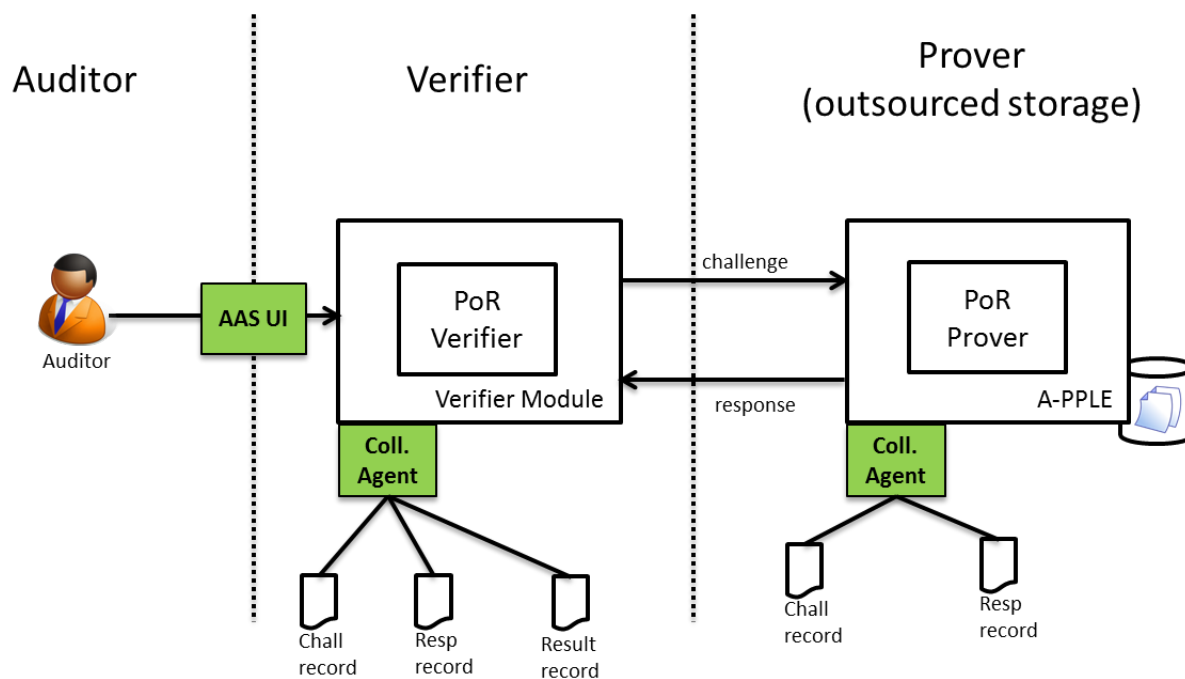


Figure 7 Audit Agent System Proof of Retrievability Integration

## 6 Interaction with the A-PPL Policy Language

As previously described, the main inputs to AAS are in the form of machine-readable policies, A-PPL. A-PPL can be used to define the details of evidence collection and evaluation. Additionally, AAS also takes additional information, an audit task configuration, as input. The configuration of audit tasks is done by an auditor using the web-based dashboard. This mainly includes the data collection sources and tools to use to collect data. This bridges the gap between policy languages not including all the necessary configuration parameters required for defining an audit task (e.g. ID of a virtual machine). To ease the audit task configuration, a cloud provider may also provide several pre-defined audit tasks to choose from and customize. AAS is capable to enable AAS to perform security audits (e.g. regarding the correct implementation of security controls) in addition to A-PPL-based audits.

Before an example audit task configuration is shown the mapping between A-PPL to AAS elements for access control and data handling is shown.

### 6.1 From A-PPL to Audit Task

Figure 8 depicts the relationship between A-PPL policies, audit policies and audit tasks. A-PPL policies describe access control rules, data usage rules and other obligations related to accountability in a machine-readable way. This is taken as a basis to create audit policies by extracting and mapping these obligations to audit tasks and tools respectively. Since this will be in some cases a semi-automatic process, input is needed from the auditor (e.g., tool configuration parameters). An audit policy is therefore a container, which includes multiple audit tasks needed to audit the adherence or violation of an A-PPL policy. Whenever possible, this is to be automated by AAS. However, we acknowledge that A-PPL policies will most likely not include information about where required evidence for (non-) compliance may be found, this information has to be complemented by the auditor. This especially includes agent- related information like the mapping of certain kinds of policies to the evidence source that may be needed.

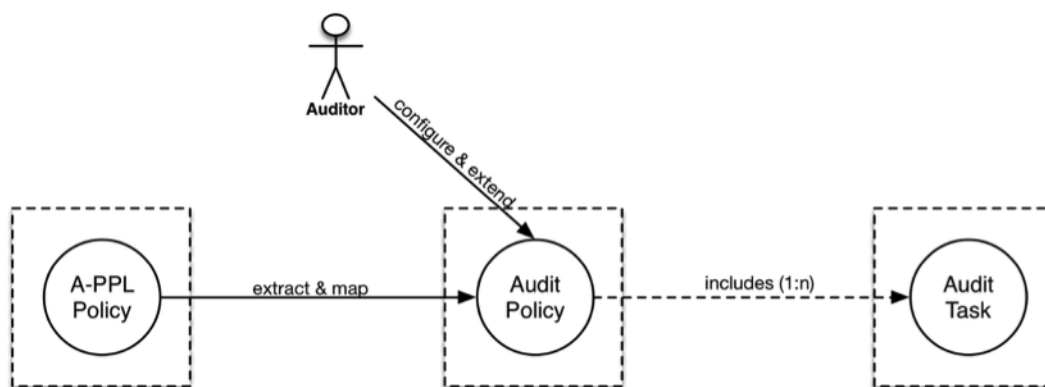


Figure 8 A-PPL to Audit Task

### 6.2 A-PPL Access Control to AAS Mapping

In Table 1 the mapping between access control policy language elements of A-PPL are mapped to data structures used in AAS. These data structures are created during the input phase in the Audit Policy Module. As can be seen, there are no direct counterparts in AAS for *Policy* and *Target*.

An A-PPL *RuleSet* is mapped to an *AuditPolicyGroup* that serves as a logical grouping element in the UI. An *Audit Policy* is directly mapped to an XACML rule. It wraps a set of *AuditTask* elements that describe evidence collection and evaluation jobs that need to be performed. Based on the results of the audit tasks, an audit result is generated for the *AuditPolicy*.



Table 1 A-PPL XACML to AAS Mapping

A-PPL XACML Element	AAS Element	Description
Policy	No direct counterpart in AAS	General information extracted (e.g. policy identifier, tenant identifier etc.)
Target	No direct counterpart in AAS	General information extracted (e.g. resource identifier)
Group of XACML Rules (RuleSet)	AuditPolicyGroup	(UI) Element for grouping AuditPolicies belonging to the same A-PPL policy
Rule (XACML)	AuditPolicy	Describes a set of checks (AuditTasks) that need to be performed to evaluate access control compliance
Action (XACML)	AuditTask	A single check that is (part of) the compliance check

### 6.3 A-PPL Data Handling to AAS Mapping

In Table 2 the mapping between data handling policy language elements of A-PPL are mapped to data structures used in AAS. These data structures are created during the input phase in the Audit Policy Module. As can be seen, there is no direct counterpart in AAS for *DataHandlingPolicy*. It merely serves as a container for obligations.

An A-PPL *ObligationSet* is mapped to an *AuditPolicyGroup* that serves as a logical grouping element in the UI. An *Audit Policy* is directly mapped to an obligation. It wraps a set of *AuditTask* elements that describe evidence collection and evaluation jobs that need to be performed. Based on the results of the audit tasks, an audit result is generated for the AuditPolicy.

Table 2 A-PPL DataHandling to AAS Mapping

A-PPL DataHandling Element	AAS Element	Description
DataHandlingPolicy	No direct counterpart in AAS	no counterpart in AAS
Group of obligations (ObligationSet)	AuditPolicyGroup	(UI) Element for grouping AuditPolicies in AAS belonging to the same A-PPL policy



Obligation	AuditPolicy	Describes a set of checks (AuditTasks) that need to be performed to evaluate obligation compliance
Action/Trigger	AuditTask	A single check that is (part of) the compliance check

#### 6.4 Other Types

In Table 3 a generic mapping between requirements that are mapped to data structures used in AAS. These data structures are created during the input phase in the Audit Policy Module.

A *group of requirements* is mapped to an *AuditPolicyGroup* that serves as a logical grouping element in the UI. An *Audit Policy* is directly mapped to a requirement. It wraps a set of *AuditTask* elements that describe evidence collection and evaluation jobs that need to be performed. Based on the results of the audit tasks, an audit result is generated for the AuditPolicy.

**Table 3 Non-A-PPL to AAS Mapping**

Non-A-PPL policy Element	AAS Element	Description
Group of requirements	AuditPolicyGroup	(UI) Element for grouping AuditPolicies belonging to the same requirement
Requirement	AuditPolicy	Describes a set of checks (AuditTasks) that need to be performed to evaluate if measures are put in place and/or if there are any possible violation during operation
Sub-requirement	AuditTask	A single check that is (part of) the compliance check

#### 6.5 Audit Task Example

Figure 9 depicts a fraction of an A-PPL policy. In this case, it is a data handling obligation dealing with data retention requirements. For each PII data set this obligation applies to, the A-PPL Engine enforces a maximum lifetime of one year for each data set before it is deleted automatically. The portions highlighted in red are the most relevant during the audit task extraction process. From the obligation with *elementId* an AuditPolicy template is extracted. This AuditPolicy template includes tasks related to data retention audits. For instance, from the trigger *TriggerAtTime* and the action *ActionDeletePersonalData* an AuditTask for data remains outside of the scope of A-PPL-E is derived (see Scenario A1 in Section 4.2.1). This task is configured information that can be extracted automatically, such as *elementId* and *Duration*, but is also configured with manually added information such as virtual resource identifiers that are audited (e.g. virtual machine names).

```

<ob:Obligation elementId="a-ppl_rule_5">
  <ob:TriggersSet>
    <ob:TriggerAtTime>
      <ob:Start>
        <ob:StartNow/>
      </ob:Start>
      <ob:MaxDelay>
        <ob:Duration>P1Y0M0DT0H0M0S</ob:Duration>
      </ob:MaxDelay>
    </ob:TriggerAtTime>
  </ob:TriggersSet>
  <ob:ActionDeletePersonalData/>
</ob:Obligation>

```

Figure 9 A-PPL Data Retention Obligation

In Figure 10 an example of this semi-automatic setup is shown. Information such as the type of AuditPolicy and available tasks (displayed by the *Data handling policies* and *Tasks* columns), retention time or the elementId (which is used behind the scenes) are taken directly from the policy, whereas the auditor is still required to provide the desired audit type, agent container, audit interval (in case of periodic audit and virtual resources identifier (in this case the virtual machine name).

Audit Agent System

Audit overviewCreate auditResultsRecords

# A4Cloud - Audit Agent System

Create audit

Data HandlingAccess controlCustom

Data handling policies

Data retention policy

extract from policy

Tasks

Snapshot check

This audit task checks if a snapshot violation occurred

EditDelete

Configuration

1. Audit type\*Periodic

2. Container\*CMS

3. Check interval:\*320000


4. VM name:\*someCimOSTests

5. Allowed existence time:\*P1Y0M0DT0H0M0S

save

☐ I have reviewed the task list and approve of the audit policy




Submit



CLOUD  
ACCOUNTABILITY  
PROJECT

A4Cloud Project  
www.a4cloud.eu

Follow us



More side info in the Footer

If you need to present some more information at the bottom, do it here.

1. Maybe in form of a list

2. This footer resizes automatically to fit the contents that you put on it

3. Footers are used mostly to present quick links to information about the service, like to the privacy policy, the contact information, etc.

Disclaimer

Your tool might need some kind of disclaimer. If your disclaimer is too long, present the important parts of it here, and provide a link to the full version in a modal dialog, or similar.

Figure 10 AAS Audit Data Retention Audit Task Semi-automatic Setup

FP7-ICT-2011-8-317550-A4CLOUD

Page 35 of 53



therefore need to be collected by the AAS. On the other hand, the A-PPL Engine is the central hub for controlling how policy violations are to be handled by the system (e.g., by reporting violations or invoking other A4Cloud tools to start further investigation).

To collect evidence from A-PPL-E's logging facility, AAS or more precisely its Collection Agents must be given access to the logs. A-PPL-E supports two different logging mechanisms:

1. File-based logging: in this case, log events are written in a log file. Any program, that has sufficient access privileges, is able to read the logs and extract information as needed. However, this is also the simplest approach to logging, which does not necessarily provide enough means of data protection.
2. Logging using Transparency Log: in this case, log events are stored in TL. Any program that needs to access log events in TL needs to be specified as a recipient because of the implicit encryption per recipient that comes with the use of TL. This approach provides several security properties to the logging mechanism.

The more preferable option is using TL as logging facility. It provides security and privacy protection mechanisms out of the box. Additionally, providing additional actors, such as the AAS collection agents, with access to log messages generated in A-PPL-E is relatively easy by adding additional message recipients in A-PPL-E.

In this particular case, AAS provides a specialized Collection Agent for A-PPL-E. The agent is configured as a recipient of log messages in the TL instance used by A-PPL-E. This is depicted in Figure 11 using the orange connection between A-PPL-E and AAS with data flowing from A-PPL-E in direction to AAS. The collection agent receives log messages from A-PPL-E and transforms them into evidence records. The records are then stored in the evidence store according to which policy they have been collected for.

The second data flow from A-PPL-E to AAS does not use TL as a means of transportation, but A-PPL-E's REST interface. Via this interface data is requested by AAS, most importantly policy documents (XML). A-PPL-E provides access to policy templates by exposing the *getPolicy* function call. The AAS core component APM implements a workflow for retrieving policies using this function call and extracting audit policies and tasks from them.

AAS uses A-PPL-E not only as a valuable source of evidence, but also a means for dispatching notifications in case of incident detection. A-PPL policies describe how policy violations shall be handled, be it either by generating notifications via a pre-defined communication channel such as TL or email or by just logging the event. AAS does not have a comparable mechanism in place. However, since AAS detects incidents and policy violations, reporting such events (i.e. sending notifications) is an important part of AAS. Policy violation notifications (as conceptually described in (Włodarczyk et al., 2015)) are generated in AAS (XML). These notifications are forwarded to A-PPL-E using its *triggerPolicyViolation* REST function call. Upon reception of such a violation notification, A-PPL-E acts according to what has been defined in the matching *Action* in the A-PPL policy.

### 7.1.2 DTMT Interfaces

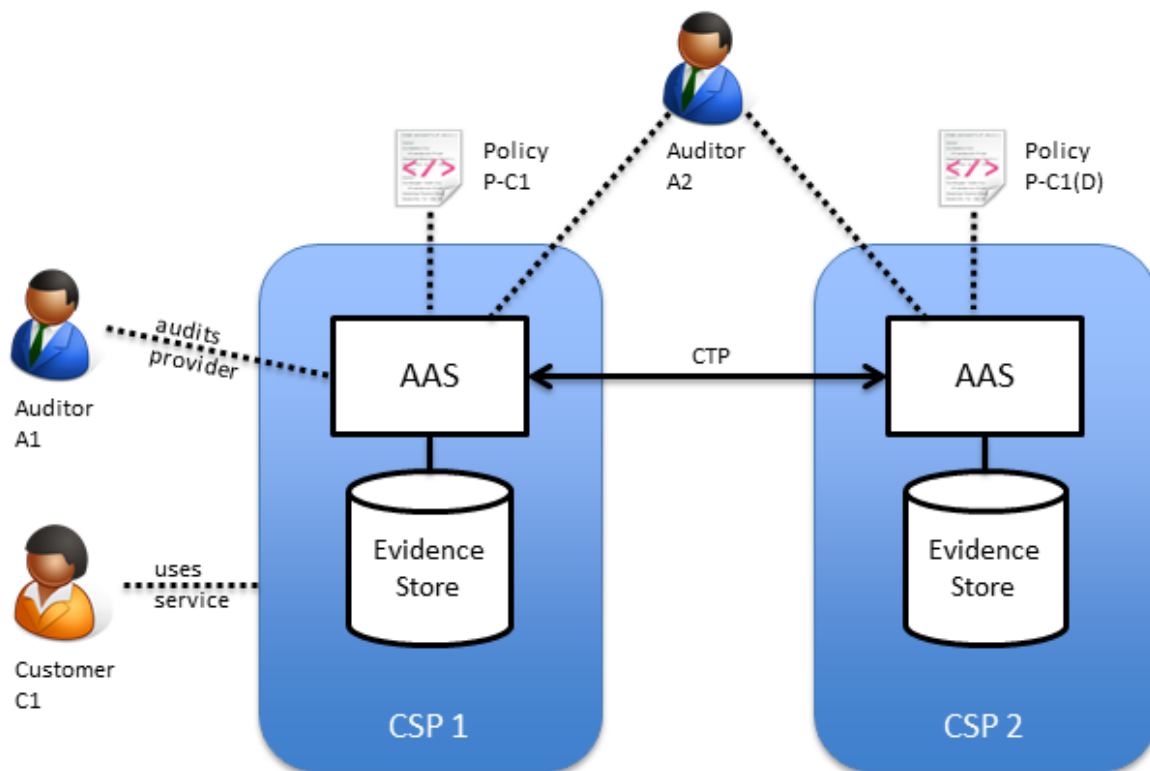
The interfacing with DTMT can be achieved similar to the both approaches described in the previous section about A-PPL-E interfacing. However, there is an additional option for this tool. In case the DTMT detects a data transfer violation on the IaaS level, it raises a violation in A-PPL-E. This makes DTMT; just as AAS, a detective tool. Therefore, the evidence produced by DTMT, in most cases, does not have to be analysed or even needs to be reacted to. When DTMT raises a violation in A-PPL-E, using the same *triggerPolicyViolation* call described in Section 7.1.1, this is typically logged in A-PPL-E. AAS can therefore pick the violation notification up in A-PPL-E, transport it back to AAS, using A-PPL-E logging as an evidence source, and store the evidence record. In cases where violations are not logged in A-PPL-E, one of the collection mechanisms (file-based or using TL) described in Section 7.1.1 are applied to DTMT. However, such cases should not happen, since we regard a missing logging obligation in the policy to be a policy configuration failure.

## 7.2 External Interfaces

In the following sections, the interfaces of AAS provided outside the scope of the A4Cloud toolset are described. Mainly, this includes the REST interface for audit task management and evidence handling provided by the AAS and the integration of the CSA Cloud Trust Protocol (CTP) (Cloud Security Alliance (CSA), 2015).

### 7.2.1 Multi-Provider Data Exchange

Figure 12 shows a simplified abstract version of the general scenario shown previously in Figure 3. Two cloud service providers (CSP1 and 2) are involved in the actual service provision. CSP1 is considered the primary service provider having a contract with customer C1, who is not aware of the involvement of CSP2 in processing his data. Since the two CSPs are distinct companies, each is running its own instance of AAS with distinct evidence stores for evidence collection and audit purposes. The data handling obligations and preferences of C1 are described in the data handling policy P-C1. This policy primarily applies to CSP1 with whom C1 has a contract. Obligations that also apply to CSP2 are defined in P-C1(D), where D indicates downstream usage of data and applicability of obligations defined in P-C1. This has several implications on the audit process in multi-provider scenarios. To demonstrate the compliance with data protection obligations, it is not sufficient to audit CSP1, but CSP2 has to be taken into account as well, if sensitive data owned by C1 is passed over to CSP2. In the following, we describe two approaches to conduct accountability audits on complex service provision chains.



**Figure 12 Cloud Trust Protocol Integration in Audit Agent System**

In the first approach, the auditor A2 is required to assess both providers individually. Based on the audit reports, he creates a unified view of compliance along the service provision chain. The combination step is not automated but performed manually by the auditor. In this scenario, all providers in the service provision need to be known to the auditor. This is mostly the case, when regulators are considered auditors or when the primary service provider performs self-audits.

In the second approach, the auditor A1 uses the AAS instance of CSP1 as an intermediary for retrieving the audit results from CSP2. In this case, the audit results from CSP2, which are based on the obligations defined in policy P-C1(D) are retrieved by CSP1's AAS using an enhanced version of the Cloud Trust Protocol (CTP). This approach enables cases, where subsequent cloud providers are unknown to the auditor (e.g., customer C1 as an external auditor) or where providers are frequently changing.

The Cloud Trust Protocol defines a protocol and interface for requesting "elements of transparency" from cloud providers. This way, evidence about data handling in the cloud can be requested. CTP leaves the implementation of evidence collection to the cloud provider and rather defines what and how of the request process for transparency elements such as configurations and data location. This makes CTP a natural fit for integration in the AAS especially in cases of cloud service chain audits or automated retrieval of audit reports.

### 7.2.2 REST Interface

The interfaces exposed by the AAS tool can be broadly categorized into being audit/AAS-specific and evidence-specific. Evidence-specific is closely related to functionality described in the Framework of Evidence (C8) and the collection and storage of evidence, whereas audit/AAS-specific provides an audit-focused layer on top of the evidence collection functionality. The API also includes the definition and evaluation of audit policies as well as the presentation of audit results. Table 4 contains a summary of the AAS REST interface broadly categorized functions related to evidence collection and audits, runtime monitoring of the AAS platform and evidence functionality.

**Table 4 REST Interface provided by AAS**

Category	API Function	Description	Exposed To
Evidence Collection and Audit Workflow	extractFromPolicy	Set the policy file to extract the audit tasks	Auditor
	setAuditTask	Definition of audit task to be performed for checking compliance with audit policy.	Auditor
	requestAuditReport	Requests the audit results of a single or all audit tasks	Auditor
Runtime Monitoring of AAS Components for User Interface	requestAuditTaskConfiguration	Requests the configuration for a given audit task	Auditor
	requestRunningAgents	Requests all agents running on the platform (i.e., all containers)	Auditor
	requestRunningAgentsByAuditTask	Requests the running agents for a given audit task	Auditor
	requestRunningAgentsByContainer	Requests the agents running in a given container	Auditor
	requestRunningAuditTasks	Requests the currently executed audit tasks	Auditor
	requestRunningContainers	Requests the running JADE agent containers (core and clients) inside an AAS platform	Auditor
Evidence-specific Functionality	requestEvidence	Requests a record, or a collection of records respectively, from the evidence repository, given an identifier of	Auditor

		the record (tenant, policyid.ruleid, timeframe).	
--	--	--	--



## 8 Scalability of Evidence Collection

In this chapter, potential bottlenecks that may impact the scalability of the framework of evidence and AAS are discussed. Additionally, mechanisms that mitigate these problems are described. Table 5 includes an overview of potential bottlenecks identified in the framework of evidence and AAS. Two general views are considered: *Non-technical* and *Technical*. The *Non-technical* view comprises of the *Management* phase. The *Technical* view comprises of the *Setup* and *Runtime* phases. At each phase possible bottlenecks are described and their mechanisms to counteract the bottleneck.

In the following sections, the bottlenecks alongside possible solutions are analysed in more depth.

**Table 5 Scalability Bottlenecks and Solutions**

	Phase	Bottleneck	Description	Mechanism
<b>Non-technical</b>	Management	Audit Plan	Ability to scale with amount of audit plans	Grouping hierarchy for tasks; Grouping by service; Automation of plan creation
		Policy	Ability to scale with amount of input policies	Grouping policies by tenants; Grouping by service; Automation of audit task extraction from policies
<b>Technical</b>	Setup	Provisioning	The provisioning and especially the need for manual configuration need to be considered in large deployments.	Lightweight client runtime framework
	Runtime	Evidence Storage	The amount of storage required for evidence depends highly on the level of detail defined in evidence collection, audit policies and tasks. Further the retention of evidence is an important factor.	Distributed storage, at least one for each tenant, but can be further scaled
		Data Transfer Volume	Collected evidence is transported from the evidence source to storage and the processing locations. Depending on the level of detail defined in evidence collection, audit policies and audit tasks, large amounts of data in terms of actual size need to be transferred over the network.	Distributed components placed as near as possible to the evidence source; Data Compression; Evidence references allow to link to evidence instead of transfer

		Message Count	Collected evidence is transported from the evidence source to storage and the processing locations. Depending on the size of the system with respect to the amount of monitored evidence sources and active audit tasks, a large amount of messages in terms of message count need to be transferred over the network.	Distributed components placed as near as possible to the evidence source; Message buffering
		Processing	Processing overhead introduced by evidence collection (load introduced on external systems) Processing overhead introduced by encryption	Load balanced distributed components
		Process Control	Ability to scale agent lifecycle management, monitoring and error handling capabilities with the amount of agents in the system	Lightweight agent interaction protocol
		Cloud Management System	The CMS is one of the most important sources of evidence. Typically, a CMS provides a monitoring or audit API to collect evidential data. The interface may not scale well with the load introduced by multiple collection instances.	Push implementation of evidence collection over poll implementation; Load balancing for monitoring nodes; Tight integration of evidence collection with message bus interface

### 8.1 Non-technical Aspects of Scalability

The Non-technical view on scalability is focused on the Management phase. In a cloud ecosystem, there are typically a lot of different tenants sharing the same resources. Also, the number of services provided by or enabled by a cloud provider can be equally big. Therefore, the framework of evidence and AAS need to scale with these numbers. From an AAS point of view, this means that an auditor needs the tool to enable him to efficiently manage a large number of audit policies for large numbers of services and many customers.

These issues are addressed in AAS by automated audit policy extraction from an input policy (see Section 6.1). The amount of manual input is thereby significantly reduced. The second mechanism for improving scalability is grouping of audit tasks for similar resources (e.g., virtual machines with identical jobs spawned from the same base template).

### 8.2 Technical Aspects of Scalability

Technical aspects of scalability have been divided into the *Setup* phase, putting the AAS into place, and a *Runtime* phase, putting the AAS into action.

#### 8.2.1 Setup Phase

The *Provisioning* bottleneck described in the *Setup* phase describes the complexity and need for manual intervention during the setup of the FoE and AAS components in a service provider's environment. As previously stated, there are a lot of potential evidence sources scattered across a cloud infrastructure. Each of these places needs to be prepared to be integrated in the FoE and AAS. The dynamic nature of cloud computing makes this a complex task that is not manageable manually. Two prime examples for this are rapid-provisioning of virtual resources and extension of the cloud infrastructure itself. In the former case, rapid creation and deletion of for example virtual machines requires the evidence collection runtime requirements to be deployable with minimal human interaction and fully automated. In the latter case, the same requirements facilitate easy extensibility of the infrastructure.

In AAS this problem is addressed by relying on Java-based software agent technology for evidence collection. In general, the only requirement for running evidence collection is the availability of a Java Runtime Environment (JRE) and the JADE runtime environment, which does not require any additional dependencies. This allows for a lightweight agent runtime environment that requires minimal amounts of configuration, which can easily be automated. The agent runtime environment may then be automatically deployed as part of the resource provisioning process or pre-installed.

### 8.2.2 Runtime Phase

*Evidence Storage*, *Data Transfer Volume* and *Message Count* refer to bottlenecks stemming from the size or frequency of evidence collection. Since evidence is generated anywhere in a cloud infrastructure, the amount of messages required to transfer the data to the storage and processing components scales with the size of the infrastructure.

- The *Message Count* increases with the amount of evidence sources and the frequency by which evidence records need to be stored.
- The *Data Transfer Volume* increases with the actual size of the payload in evidence records.
- *Evidence Storage* increases over time depending on the data retention requirements imposed on the *Evidence Store* (i.e., how long do evidence records need to be saved before they can be discarded).

All three can to some degree be addressed by the introduction of compression, buffering, evidence referencing and distribution of components.

- Most evidence collection resources are log files. These log data are typically very suitable for compression, which effectively reduces the size of data that needs to be transferred.
- Buffering evidence records (by number of records and after a specific timeout is reached) at the collector and sending them in larger packets (ideally compressed) effectively reduces the amount of messages in the network compared to sending records as soon as they are available.
- Evidence referencing allows storing a reference (along with a cryptographic hash for integrity verification) to a specific piece of evidence in a record (URI) instead of the actual data itself. This mechanism is particularly effective when evidence is of reasonably large size and is unlikely to change in a longer period of time, for example a certificate, an annual report or an audit report (actual document).
- The distribution of components allows for an easy load balancing in the collection, storage and processing components. Since all of them are implemented as agents, they can either be moved to a location, where sufficient processing resources (e.g., CPU and storage) are available or new instance for example of the Evidence Store can be spawned to lessen the load on existing instances. Also, the flow of information can be optimized. Two different processing flows in AAS have previously been described in Section 5.1.4. By moving the processing agent closer to the evidence collector, a shortcut is implemented, that lessens the load on the network both in message count and in data transfer volume.

The *Processing* bottleneck describes to additional computational resources required for evidence collection, encryption and processing when implementing AAS in a cloud infrastructure. Furthermore, evidence collection may introduce additional load on existing systems that are now used for evidence collection.

The Audit Agent Controller is an important centralized component in AAS. It's the agent that is primarily used for *Process Control*. *Process Control* describes all actions in AAS that are required for the agent lifecycle management. These actions include: (i) instantiation according to a policy, (ii) agent migration in the distributed environment, (iii) agent destruction, (iv) agent monitoring, (v) error and (vi) exception handling.

Agent monitoring and error and exception handling are the actions with the biggest impact with respect to load generated on the network. AAS addresses these issues by using a very light-weight messaging protocol based on the JADE-provided Agent Communication Language (ACL) (JADE, 2015). Also, since the main process of evidence collection, storage and processing does not require communication the central AAC component, the amount of required communication between agents is reduced even more.

*Cloud Management System* includes the cloud management system's monitoring and audit interfaces. When these interfaces need to serve requests by evidence collection agents, it is important to choose mechanisms that are light on the processing side (i.e., push data to the collection agent instead of polling). Furthermore, a tight integration with the CMS may be beneficial (e.g., an evidence collector as an additional listener on OpenStack's message bus), both with respect to a greater level of detail exposed to the evidence collection as well as performance optimizations. These principles should also be applied to any other complex system with external interfaces that can be used for evidence collection (e.g., hardware monitoring systems).

## 9 Privacy Analysis of the Audit Agent System

C-7 WP has conducted an initial privacy analysis of the Audit Agent System using a systematic structure adapted from privacy impact assessment to evaluate the tool for privacy concerns. This sub section summarizes the results of the privacy analysis and the privacy by design guidelines provided by C-7 WP (A4Cloud, 2014).

In the privacy analysis phase, the specification of the AAS is summarised and analysed with respect to the data protection and privacy principles. The summary includes the types of personal data processed by the AAS, purposes for processing, the data retention period and the primary stakeholders of AAS. These primary privacy-related characteristics facilitate the understanding of the information flow that occurs during the life cycle of the data. The objective of the analysis of the information flow during the life cycle of the data is to answer the following questions and eventually to discern the privacy concerns.

1. How the data were collected and what are the implications, i.e., to ascertain that the agents collect only the evidences relates to the subjects it serves.
2. How the personal data were stored, transferred and communicated between the different entities in the tool's architecture.
3. How long the personal data were stored, and are the stored data anonymised
4. Where the personal data stored and the level of the storage isolation.

What prevents the agents from collecting personal data or business confidential data of another subject is of paramount importance because it results in a weakest link scenario. Compromise of one agent either directly or indirectly leads to the compromise of all the data accessible by the agents. And securing the communication links among the entities of the system is equally important which otherwise reveal considerable amount of sensitive information like when/how often a data subject's data was audited, etc.

Having identified the privacy concerns, the privacy requirements as stated in the C:7 deliverable (A4Cloud, 2014) for the AAS are

1. Minimise the impact of agents as far as possible, in particular by requiring strong authentication for defining audit tasks and for where they can execute.
2. Minimise the amount of data stored in the evidence store, protect such information from unauthorized access and delete it as soon as possible.
3. Only enable access to data in the evidence store that is strictly needed
4. Keep reports private. Only include raw data from the evidence store when it is in the interest of the data subject.
5. Secure the communication channels.
6. Ensure that all actions in the system can be traced in case of conflict.

### 9.1 Threat model and Risks

In the threat model for the AAS, the CSP is considered as an honest but curious adversary who performs the deployment and processing of the agents appropriately however interested in discovering the objective of the audit, and the content of the evidence store of other CSPs that are in the service chain. Similarly, auditors are also considered as honest but curious but the difference is, the auditors are bounded by the amount of information they can access.

### 9.2 Privacy Enhancing Technologies for AAS

As part of the privacy analysis, the C: 7-work package had suggested privacy enhancing mechanisms for AAS,

In general, all the communication links over the network should be protected by TLS with mutual authentication. Specially, to authenticate the auditors, multi-factor authentication using TLS client

certificates together with a password for each auditor is recommended alongside a good support for key management.

To address the honest but curious CSPs, the evidence store should include the privacy preserving primitives and all the information in the evidence store need to be encrypted at rest. Above all a strong access control should be enforced to access the evidence store, for this reason, Transparency Log (TL) component of the A4Clouds tools is suggested as the evidence store, it provides the required privacy and security primitives. Additionally, privacy preserving delegated word search, which relates to the evidence store and allows the auditors to query parts of the evidence store to generate cryptographic proofs could be included, the primary feature of this primitive is neither the evidence store (TL) nor the external entities can learn the content and result of this query. Another concern with respect to the honest but curious CSP adversary model is, the runtime environment where the software agents run, is operated on the very cloud service provider environment that is the accountable for its actions. This leads to the question of how trustworthy are the collected evidence which can be addressed by Accountability and Privacy Enforcement tool and Transparency Log tool.

To address the semi-trusted auditors threat model, all the audit tasks, the reports generated by the audit tasks, details of the sent out audit reports need to be logged and secured using secure logging or Transparency Log component, and this can be served as a transcript of all events in AAS and allows to resolve disputes. Furthermore, auditors have limited access to the cloud infrastructure through the Audit Agent System and have only access to collect evidence based on predefined policies which minimizes the exposure of personal data and business confidential data.

For a detailed description of the privacy analysis conducted for AAS, readers are recommended to refer to D:C:7.2 : Privacy Design Guidelines for Accountability Tools (A4Cloud, 2014) deliverable by C:7 WP of A4Cloud.

Additionally, we suggest that there are few manual checks that need to be conducted to ensure the integrity of the audit outcomes. Firstly, as noted in the privacy analysis the integrity of the agent runtime environment needs to be assured, e.g. by checking if the CSP has conducted a vulnerability analysis to its system architecture. Secondly, ensure the logs accurately reflect the actual activities of the system and that the Data Controller (DC) does not influence the outcome of the audit in any way. This can be done, e.g. by checking the configuration settings of the logging components to make sure that the logging facility is configured appropriately and also by manually checking the design documents of the architecture to ensure that every system components has an ability to log. We reckon that these manual checks are important and recommend that they need to be checked prior to the semi-automated audit tasks performed in AAS.

## 10 Conclusions

This deliverable presented the work WP-C8 Task 8.6, which focussed on the construction of a service for automated evidence collection and processing. The implementation of the principles and mechanisms described in the framework of evidence is done in the Audit Agent System (AAS). AAS is a tool, that enables cloud providers to follow an evidence-based approach to accountability by collecting evidence on what happens in their infrastructure, how data is processed and by whom. Furthermore, AAS provides auditors the capability to conduct automatic audits and monitoring of cloud service infrastructures, including automated incident detection and incident notification. Therefore, AAS plays an important role in the detective phase of the accountability life-cycle and provides input about incidents directly or indirectly to or via the A-PPL Engine tool.

The Audit Agent System leverages software agent technology at its core, to address the heterogeneity of evidence sources in a cloud infrastructure as well as to address scalability requirements stemming from distributed evidence collection and processing. It is implemented as a RESTful service using JAVA and the JAVA Agent Development Environment JADE. To address security and privacy requirements imposed by the collection of potentially confidential evidence, Transparency Log was integrated as a means for secure and privacy-friendly storage and data transfer mechanism. Furthermore, AAS incorporates A-PPL policies as the primary input for defining evidence collection, resource monitoring and audit.



## 11 References

- A4Cloud. (2014). D:C-7.2 Privacy Design Guidelines for Accountability.
- Accorsi, R. (2008). Automated Privacy Audits to Complement the Notion of Control for Identity Management. In E. de Leeuw, S. Fischer-Hübner, J. Tseng, & J. Borking (Eds.), *Policies and Research in Identity Management* (Vol. 261, pp. 39–48). Springer US. Retrieved from [http://dx.doi.org/10.1007/978-0-387-77996-6\\_4](http://dx.doi.org/10.1007/978-0-387-77996-6_4)
- Accorsi, R., & Stocker, T. (2008). Automated Privacy Audits Based on Pruning of Log Data. In *Enterprise Distributed Object Computing Conference Workshops, 2008 12th* (pp. 175–182). <http://doi.org/10.1109/EDOCW.2008.18>
- Azraoui, M., Elkhiaoui, K., Molva, R., & Önen, M. (2014). StealthGuard: Proofs of Retrievability with Hidden Watchdogs. In M. Kutyłowski & J. Vaidya (Eds.), *Computer Security - ESORICS 2014* (Vol. 8712, pp. 239–256). Springer International Publishing. Retrieved from [http://dx.doi.org/10.1007/978-3-319-11203-9\\_14](http://dx.doi.org/10.1007/978-3-319-11203-9_14)
- Blažič, A. J., Klobučar, T., & Jerman, B. D. (2007). Long-term trusted preservation service using service interaction protocol and evidence records. *Computer Standards & Interfaces*, 29(3), 398–412. <http://doi.org/10.1016/j.csi.2006.06.004>
- Boyens, J., Paulsen, C., Moorthy, R., Bartol, N., & Shankles, S. (2013). Supply Chain Risk Management: Practices for Federal Information Systems and Organisations. NIST SP 800-161.
- Brandner, R., Pordesch, U., & Gondrom, T. (n.d.). Evidence Record Syntax (ERS). Retrieved May 11, 2014, from <http://tools.ietf.org/html/rfc4998>
- Butin, D., Chicote, M., & Métayer, D. L. (2013). Log Design for Accountability. In *IEEE Symposium on Security and Privacy Workshops* (pp. 1–7). IEEE Computer Society. Retrieved from <http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6564486>
- Cederquist, J. G., Corin, R., Dekker, M. a. C., Etalle, S., Hartog, J. I., & Lenzini, G. (2007). Audit-based compliance control. *International Journal of Information Security*, 6, 133–151. <http://doi.org/10.1007/s10207-007-0017-y>
- Cloud Security Alliance. (2013). The Notorious Nine Cloud Computing Top Threats in 2013. Retrieved from

- [https://downloads.cloudsecurityalliance.org/initiatives/top\\_threats/The\\_Notorious\\_Nine\\_Cloud\\_Computing\\_Top\\_Threats\\_in\\_2013.pdf](https://downloads.cloudsecurityalliance.org/initiatives/top_threats/The_Notorious_Nine_Cloud_Computing_Top_Threats_in_2013.pdf)
- Cloud Security Alliance (CSA). (2014). CSA Privacy Level Agreement Europe, v.2.
- Cloud Security Alliance (CSA). (2015). *Cloud Trust Protocol*. Retrieved from <https://cloudsecurityalliance.org/research/ctp>
- CVE-2011-3389. (2011). BEAST CVE-2011-3389. Retrieved from <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-3389>
- CVE-2014-0160. (2014). Heartbleed CVE-2014-0160. Retrieved from <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160>
- CVE-2014-3566. (2014). POODLE CVE-2014-3566. Retrieved from <https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3566>
- D'Errico, M., & Pearson, S. (to appear). Towards a Formalised Representation for the Technical Enforcement of Privacy Level Agreements. In *Proceedings of the IEEE 1st International Workshop on Legal and Technical Issues in Cloud Computing (CLaw)*.
- ENISA. (2009). Cloud Computing – Information Assurance Framework. European Network and Information Security Agency (ENISA).
- Etalle, S., & Winsborough, W. H. (2007). A Posteriori Compliance Control. In *In Proc. 12th ACM Symposium on Access Control Models and Technologies*. ACM Press.
- Felici, M., & Pearson, S. (2014). D:C-2.1 Report detailing conceptual framework.
- Gittler, F., & Koulouris, T. (2014). D:D-2.2a: High-level architecture.
- JADE. (2014). *Java Agent DEvelopment framework*. Retrieved from <http://jade.tilab.com>
- JADE. (2015). *JADE Agent Communication Language (ACL)*. Retrieved from <http://jade.tilab.com/doc/api/jade/lang/acl/package-summary.html>
- Kavanagh, K. M., Nicolett, M., & Rochford, O. (2014). Magic Quadrant for Security Information and Event Management. Gartner.
- Lee, W., & Stolfo, S. J. (1998). Data Mining Approaches for Intrusion Detection. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7* (pp. 6–6). Berkeley, CA, USA: USENIX Association. Retrieved from <http://dl.acm.org/citation.cfm?id=1267549.1267555>
- Lundin, E., & Jonsson, E. (2002). Survey of intrusion detection research (Technical Report). Chalmers University of Technology.

- Lunt, T. F. (1988). Automated Audit Trail Analysis and Intrusion Detection: A Survey. In *Proceedings of the 11th National Computer Security Conference* (pp. 65–73).
- Nimity. (2014). Privacy Management Accountability Framework. Nimity.
- Patcha, A., & Park, J.-M. (2007). An Overview of Anomaly Detection Techniques: Existing Solutions and Latest Technological Trends. *Comput. Netw.*, 51(12), 3448–3470.  
<http://doi.org/10.1016/j.comnet.2007.02.001>
- Pearson, S., & Yee, S. (Eds.). (2013). Enterprise Information Risk Management: Dealing with Cloud Computing. In *Privacy and Security for Cloud Computing*.
- Pulls, T., & Martucci, L. (2014). D:D-5.2 User-Centric Transparency Tools V1 (Technical Report).
- Pulls, T., & Peeters, R. (2015). *Balloon: A Forward-Secure Append-Only Persistent Authenticated Data Structure*.
- Pulls, T., Peeters, R., & Wouters, K. (2015). *Distributed privacy-preserving transparency logging*.
- Ruebsamen, T., Pulls, T., & Reich, C. (to be published). Secure Evidence Collection and Storage for Cloud Accountability Audits. In *Proceedings of the 5th International Conference on Cloud Computing and Services Science*. Lisbon, Portugal: SciTePress 2015.
- Ruebsamen, T., & Reich, C. (2013). Supporting Cloud Accountability by Collecting Evidence Using Audit Agents. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on* (Vol. 1, pp. 185–190). <http://doi.org/10.1109/CloudCom.2013.32>
- Santa de Oliveira, A., Sendor, J., ThànhPhúc, V., Vlachos, E., Azraoui, M., Elkhyaoui, E., ... D'Errico, M. (2015). D:D-3.2 Prototype for accountability enforcement tools and services (Technical Report).
- Schatz, B., & Clark, A. J. (2006). An open architecture for digital evidence integration. Retrieved from <http://eprints.qut.edu.au/21119/>
- Turner, P. (2005). Unification of digital evidence from disparate sources (digital evidence bags). *Digital Investigation*, 2(3), 223–228.
- Wang, C., & Zhou, Y. (2010). A collaborative monitoring mechanism for making a multitenant platform accountable. *Proc. HotCloud*. Retrieved from [https://www.usenix.org/legacy/event/hotcloud10/tech/full\\_papers/WangC.pdf](https://www.usenix.org/legacy/event/hotcloud10/tech/full_papers/WangC.pdf)
- Włodarczyk, T. W., Pais, R., Agrawal, B., Molland, H., Gulzar, H., Rübsamen, T., ... Royer, J.-C. (2014). D:C-8.1 Framework of Evidence (Technical Report).

Włodarczyk, T. W., Pais, R., Rübsamen, T., Reich, C., Azraoui, M., Pulls, T., ... Felici, M. (2015). D:C-8.2 Framework of Evidence (Technical Report).

Yao, J., Chen, S., Wang, C., Levy, D., & Zic, J. (2010). Accountability as a service for the cloud. In *Services Computing (SCC), 2010 IEEE International Conference on* (pp. 81–88). IEEE.

Retrieved from [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=5557218](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5557218)

## 12 Index of figures

Figure 1 Accountability Framework .....	8
Figure 2 Mapping of Framework of Evidence on Audit Agent System.....	15
Figure 3 Reference Scenario Overview (Demonstrator) .....	18
Figure 4 AAS Architecture Overview .....	25
Figure 5 Approaches for Automated Policy Violation Detection in AAS .....	26
Figure 6 PoR setup (above) and PoR audit (below) .....	28
Figure 7 Audit Agent System Proof of Retrievability Integration .....	30
Figure 8 A-PPL to Audit Task .....	31
Figure 9 A-PPL Data Retention Obligation.....	34
Figure 10 AAS Audit Data Retention Audit Task Semi-automatic Setup .....	35
Figure 11 AAS Interfaces with A4Cloud Toolset (Internal).....	36
Figure 12 Cloud Trust Protocol Integration in Audit Agent System.....	38

## 13 Index of tables

Table 1 A-PPL XACML to AAS Mapping.....	32
Table 2 A-PPL DataHandling to AAS Mapping .....	32
Table 3 Non-A-PPL to AAS Mapping .....	33
Table 4 REST Interface provided by AAS.....	39
Table 5 Scalability Bottlenecks and Solutions.....	41