future internet

MDPI

Article

Identity-as-a-Service: An Adaptive Security Infrastructure and Privacy-Preserving User Identity for the Cloud Environment

Tri Hoang Vo [1,3,*], Woldemar Fuhrmann [1], Klaus-Peter Fischer-Hellmann [2] and Steven Furnell [3,4]

[1] Department of Computer Science, University of Applied Sciences Darmstadt, 64295 Darmstadt, Germany; w.fuhrmann@fbi.h-da.de
[2] Digamma GmbH, 64367 Darmstadt, Germany; K.P.Fischer-Hellmann@digamma.de
[3] Centre for Security, Communications & Network Research, University of Plymouth, Plymouth, PL4 8AA, UK; S.Furnell@plymouth.ac.uk
[4] Centre for Research in Information and Cyber Security, Nelson Mandela University, Port Elizabeth, South Africa; S.Furnell@plymouth.ac.uk
* Correspondence: vohoangtri@gmail.com

Received: 12 March 2019; Accepted: 8 May 2019; Published: date

Abstract: In recent years, enterprise applications have begun to migrate from a local hosting to a cloud provider and may have established a business-to-business relationship with each other manually. Adaptation of existing applications requires substantial implementation changes in individual architectural components. On the other hand, users may store their Personal Identifiable Information (PII) in the cloud environment so that cloud services may access and use it on demand. Even if cloud services specify their privacy policies, we cannot guarantee that they follow their policies and will not (accidentally) transfer PII to another party. In this paper, we present Identity-as-a-Service (IDaaS) as a trusted Identity and Access Management with two requirements: Firstly, IDaaS adapts trust between cloud services on demand. We move the trust relationship and identity propagation out of the application implementation and model them as a security topology. When the business comes up with a new e-commerce scenario, IDaaS uses the security topology to adapt a platform-specific security infrastructure for the given business scenario at runtime. Secondly, we protect the confidentiality of PII in federated security domains. We propose our Purpose-based Encryption to protect the disclosure of PII from intermediary entities in a business transaction and from untrusted hosts. Our solution is compliant with the General Data Protection Regulation and involves the least user interaction to prevent identity theft via the human link. The implementation can be easily adapted to existing Identity Management systems, and the performance is fast.

Keywords: identity-as-a-service; federated identity management; privacy-preserving; purpose-based encryption; purpose-based access control; attribute-based encryption; cloud adaptation; cloud migration

1. Introduction

In a local hosting environment, traditional applications have their own implementations for authentication and authorisation. Each application has *n* user accounts. *Personal Identifiable Information* (PII) or *user identity* is information about a person (e.g., name, age, addresses), which makes it possible to identify him or her [1]. The management process of PII associated with different levels of access control gave birth to Identity Management (IDM) [1]. In one security domain, PII may be stored in a central Identity Provider (IdP) and disseminated to Service Providers (SPs) on

demand. An SP may require PII to authorise a user request, to complete a business transaction, or to customise its service [2].

In federated IDM, SPs from different security domains exchange messages containing authentication and authorisation credentials about users [3]. As a result, federated IDM reduces the cost for SPs because they no longer manage users that are not under their control [4]. SPs form a federation by developing offline operating agreements [4]. For instance, they may specify which PII they want to exchange, which protocol to use (e.g., Security Assertion Markup Language (SAML) [5], or Web Services Federation [6]), and whether the client may act on behalf of a user to access the service. In the end, SPs expect each other to behave accordingly and thus they establish a trust relationship [7]. Afterwards, developers adapt the implementation of Authentication and Authorisation Infrastructure (AAI), depending on these agreements [4].

In cloud computing, SPs come from various security domains; provide themselves as cloud services and may cooperate with each other. From the beginning, they may establish their trust manually. However, as time goes by, cloud services may migrate to another Software-as-a-Service (SaaS) cloud provider and may adapt their trust upon each change [8]. Figure 1 shows various motivation scenarios. In the first scenario (Figure 1a), a shopping service consumes a delivery service in a business-to-business (B2B) scenario. Each time the delivery service migrates from a local host to another cloud provider, developers have to adapt their security infrastructures again manually. As an Independent Service Vendor running on many platforms, cloud services may refuse to change their application implementations to adapt to a given one [8].

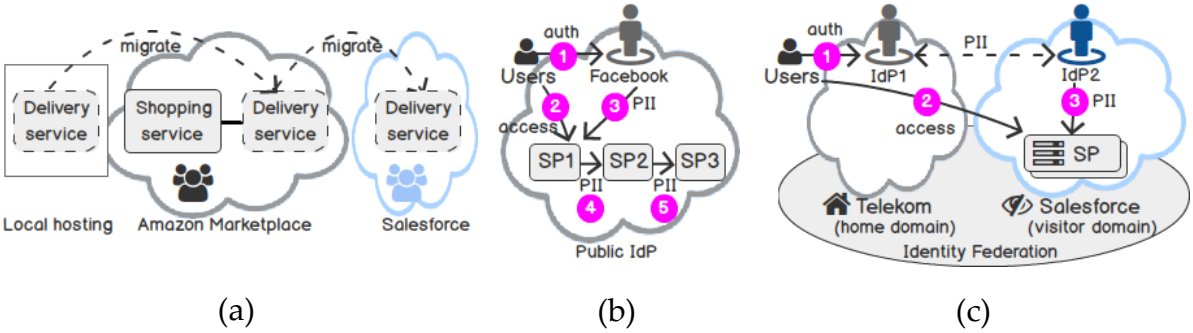


Figure 1. Motivation scenarios. (**a**) Cloud migration; (**b**) Identity propagation; (**c**) Identity federation. Note: Throughout this paper, we will use Salesforce, Facebook, and Telekom in our examples since they represent familiar examples, but we note that our examples are not tied specifically to these organisations and our concepts are equally applicable to other vendors and services.

On the other hand, users may store their PII in the cloud environment so that cloud services may access and use it on demand. From the user's perspective, users may prefer to access multiple cloud services in federated security domains, but also preserve their privacy [8]. Even if cloud services specified their privacy policies, it is not possible to guarantee that they follow their policies and will not (accidentally) transfer user data to another party [8]. In Figure 1b, Facebook is a public IdP that collects PII about users. After a user authenticates to Facebook, he can access an application in this domain (step 1 and 2). Facebook may also disseminate user identity to the application on demand (step 3). According to the Facebook data scandal in early 2018 [9], an application was allowed to collect PII of 50 million users for "academic" use but gave the collected data further to a company, Cambridge Analytica, for "analysis" purpose. This example shows that users typically disclose their identities with an SP (e.g., SP1) over the frontend. In the backend, SP1 may consume another service (e.g., SP2) in a B2B relationship (step 4). SP1 may be dishonest or accidentally forward PII to SP2 without user control. This scenario raises a question: how do we support identity propagation between intermediaries but also protect unauthorised access at the same time?

In the third scenario (Figure 1c), a company (e.g., Deutsche Telekom) offers its employees several applications hosted by an SaaS (e.g., Salesforce). Cloud services on Salesforce may require user identity exclusively. Thus, it may be necessary to provision user identity from Telekom to Salesforce, while keeping the authentication credential (e.g., username and password) locally at

Telekom. After a user authenticates successfully at Telekom (step 1), he can access cloud services at Salesforce (step 2). This is a typical solution for a local company that uses a SaaS provider [10]. Thanks to identity federation, cloud services can query PII from the IdP of Salesforce without contacting Telekom and gain performance (step 3). The disadvantage is that Telekom must completely trust and delegate the control for disclosing its employee's data to Salesforce. This scenario raises a question: how do we protect PII when we disseminate it from a trusted domain to an *honest-but-curious* one in identity federation? It means Salesforce may follow the protocol and computation that Telekom delegates to it, but may try to learn more information about the stored data. Even if the two companies specify privacy policies in their business contract, we cannot prevent a malicious host, malware, or an insider attack. Indeed, attackers have stolen 1.5 million personal data from the health authority in 2018 because the IdP had malware injected in it [11]. This scenario raises a question: how do we protect PII from an untrusted host?

2. The Need for Identity-as-a-Service

In e-commerce, SPs demand highly secure and flexible access control mechanisms for identity federation. However, this security feature is not a core competency [12]. Therefore, cloud services may prefer outsourcing AAI to a third party so that they can focus on developing their core business functionalities [12]. In such cases, their security infrastructures can be strengthened by a specialised provider to reduce their cost [12]. Migration of existing applications from a local hosting to a cloud provider requires substantial adaptation effort in individual architectural components [13]. A survey of cloud topology and orchestration from 2012 to 2017 identified 91 research efforts, but very few papers have addressed the security aspects [14]. Existing work has focused on migrating application components with functional and non-functional aspects [15–18]. However, none of them has focused so far on the adaptation of the required security infrastructures.

On the other hand, in the past 10 years, many efforts have been taken in protecting PII [19–29]. These approaches target certain issues but still have limitations. For instance, users require interacting with the SPs over the frontend, they neither protect identity propagation between intermediaries nor against an untrusted host. It is worth mentioning that human-PC link is the weakest link compared to the links between the PC, SP, and IdP [3]. Therefore, the human link is the major target of identity theft [3]. For this reason, we consider reducing human interaction from identity disclosure as much as possible.

The open standards organisation, OASIS, mentioned Identity-as-a-Service (IDaaS) as "an approach to Identity Management in which an entity (individual or organisation) relies on a cloud service provider that allows the entity to perform an electronic transaction, which requires identity data managed by this provider" [30]. Since the definition is non-standard and coarse, in [8] we revisited the requirements of a traditional IDM system [31] and specify which requirements are still missing in cloud computing. Furthermore, an IDM system is much more likely to succeed when it benefits all parties, including users, SPs and IdP [32]. This was our goal when we proposed two new requirements for IDaaS bringing benefits to both SPs and users: IDaaS adapts an automated trust between cloud services on demand and protects the confidentiality of PII in federated security domains [8]. Figure 2 illustrates a brief overview of our novel architecture design. First, IDaaS adapts the AAI implementation so that cloud services from various security domains can trust each other on demand. Second, a user Bob encrypts his PII in his home IDaaS, which further disseminates the ciphertext on his behalf to a given security domain before he accesses cloud services in this domain.

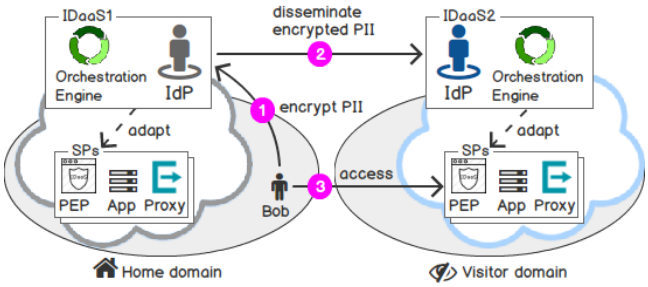


Figure 2. Identity-as-a-Service proposed model.

The main contributions of our work can be summarised as follows:

1. *Trust adaptation:* In [33], we proposed to separate the security infrastructure from the application business logic as a new virtualization layer. In particular, we move the trust and protection relationship as well as the identity propagation between application components out of the application implementation and model them as a security topology. The topology describes the AAI components, their relationships with the application components in a platform-independent modelling language. Using the topology, IDaaS can adapt the security infrastructure, establishes a dynamic trust relationship between cloud services on demand, and evaluates any changes in the runtime environment to conform to the model.
2. *Purpose-based Encryption:* In [34], we proposed a broader solution to solve the privacy issues above: We protect PII over an *intended channel* (i.e., a frontend service), as well as an *unintended channel* (i.e., a backend service is completely transparent to the user), and against an untrusted host. We also notice that absolute protection does not exist. Thus, we seek an efficient solution that is compliant with the law, in particular, the General Data Protection Regulation (GDPR) [35]. As a result, we proposed a new approach to *Purpose-based Encryption* (PBE). To the best of our knowledge, our work is the first approach to combine *Purpose-based Access Control* and *Attribute-based Encryption* (ABE) to protect the confidentiality of disseminated data with multi-authorities support. Briefly, a user Bob encrypts his PII with a disclosure policy based on three main factors: "domains", "time", and "purposes". Then, IDaaS disseminates Bob's encrypted PII in various domains. Only services hosted in specific domains (e.g., Facebook, Salesforce) can decrypt the ciphertext within a given period (e.g., 14 days), if and only if the access purposes of the service (e.g., marketing, purchase) satisfy the intended purposes that Bob specifies before. Our PBE is efficient: First, our solution involves the least user interaction to prevent identity theft via the human link. Second, the implementation can be easily adapted to existing IDM systems, and the performance is fast.

In this paper, we present the experiment results and evaluation of the above concepts. We organise the paper as follows. First, we discuss the limitations of related work. Then, we describe our novel architecture design of IDaaS in Section 4. Afterwards, we dive deeper into the trust adaptation and PBE in Sections 5 and 6, respectively. In the end, we summarise and discuss future work.

3. Related Work and Discussions

In the following sections, we discuss related work in AAI (Section 3.1) and privacy-preserving user identity (Section 3.2).

3.1. Authentication and Authorisation Infrastructure

Existing solutions for IDaaS (e.g., One Login [36], Ping Identity [37]) provide a central IdP for managing users. Their users have Single Sign-On access to cloud services, which were manually adapted to the IdP before. However, automated trust negotiation between cloud services and privacy-preserving user identity are not supported. On the other hand, several solutions from the industry provide frameworks for developers to implement AAI but lock an AAI implementation to a

vendor-specific application server or an IdP [33]. Therefore, developers have to relearn and implement repetitive tasks using APIs from different vendors. These proprietary APIs also limit the portability and interoperability of cloud services [33]. Our approach provides a holistic approach to adapt AAI for cloud services in multiple cloud providers.

Work in [38] uses Aspect-Oriented Programming to enforce authentication and authorisation on critical points of an application program (i.e., a component, a class, or a class method). However, administrators may have difficulties to define security policies by looking at the class methods. In addition, this approach adds additional performance overhead on the secured resources [38]. In comparison to our work, we only secure the public APIs of the application component. However, we change the security implementation outside the application component without affecting the running application. As a result, we increase the performance, because the Web server now performs the security-related tasks [39]. Also, we let the security architect define a flexible topology for his application (i.e., he can choose a central Policy Enforcement Point (PEP) to protect multiple application components or a dedicated PEP for a target component).

In a standard approach, the specification WS-Policy [40] and WS-SecurityPolicy [41] define frameworks for Web services to express their security constraints and requirements. However, they do not specify whether the client may call the Web service with the client identity or impersonate the original user. In such cases, the resource owner of the call is unidentified. Without identifying the resource owner, the Web service cannot authorise and audit a request correctly. Also, the policy specifies the security constraints for a platform-specific implementation only. Our approach fixes these missing gaps by modelling the trust relationships between Web services in a platform-independent model. When the security infrastructure is adapted to a target hosting, the adaptation process generates a WS-SecurityPolicy description that conforms to a trust instance.

In cloud migration, existing work has focused on migrating application components with functional and non-functional aspects [15–18] (e.g., replace a database with a Database-as-a-Service). Our approach supports legacy applications by considering them as *unprotected* components (i.e., components with no security constraints). Then, IDaaS secures them during the cloud migration. However, a security architect supports the adaptation process by describing the topology views of the AAI components.

Finally, we also need to evaluate the deployment of AAI in a target hosting. In general, a trusted third party may assert certifications for the deployment by using a Trusted Platform Module [42], a monitoring-based [43], or a test-based certification [44]. In this paper, we follow the test-based certification by defining an integration test for each trust capability. At runtime, IDaaS performs the corresponding test to check, whether a deployment conforms to the trust capability.

3.2. Privacy-Preserving User Identity

Recently, the consent receipt specification [19] defines a record of authority when a user discloses his PII to an SP. This approach involves the user granting permissions for each SP explicitly. On the other hand, anonymity approaches [20–24], aggregated Zero Knowledge Proofs [25–27], and group signatures [28,29] preserve complete anonymity for the user in a transaction with an SP. They require users to interact with an SP over the front-end. Therefore, they overload the IDM tasks to the user's decision in every transaction and do not support identity propagation over the unintended channel. Out of the methods mentioned, our approach involves the least user interaction. We require the user to select the intended purposes for his PII when he registers it to the system at the beginning. Afterwards, the authorisation is based on the defined purposes.

In contrast to an anonymity approach, the confidentiality approaches in [45–52] do not protect user anonymity but disclose PII to an entity if a disclosure policy is satisfied. Mont *et al.* [46] encrypt user data with *Identity-based Encryption* [53]. If an SP wants to decrypt the data, it has to interact with a Trusted Third Party (TTP) to provide its policies and platform configurations. Our approach does not require SPs sending their configurations to the TTP. We borrow their idea to encrypt the data with the disclosure policies, but we use ABE as an encryption scheme with a more expressive policy.

Active bundles [48] are mobile agents that use disclosure policies to authorise requests from the host and disclose the data to the host. However, the execution of a mobile agent's plaintext code in an untrusted host is not secure [54]. Recent work also took advantage of *Blockchain* technology to store data (e.g., an authentication claim [23], or an access control list [24]) in a Blockchain network. However, PII is stored in an external database. While Blockchain brings an auditable history to the stored data, it cannot protect PII from an untrusted host of the external database and the authorisation server. Our solution avoids the above issues of active bundles and Blockchain because our authorisation does not rely on the execution of plaintext code. Instead, the cryptographic computing itself performs the authorisation. If the host changes the disclosure policies (e.g., the access control list) or the cryptographic computing, the decryption simply fails.

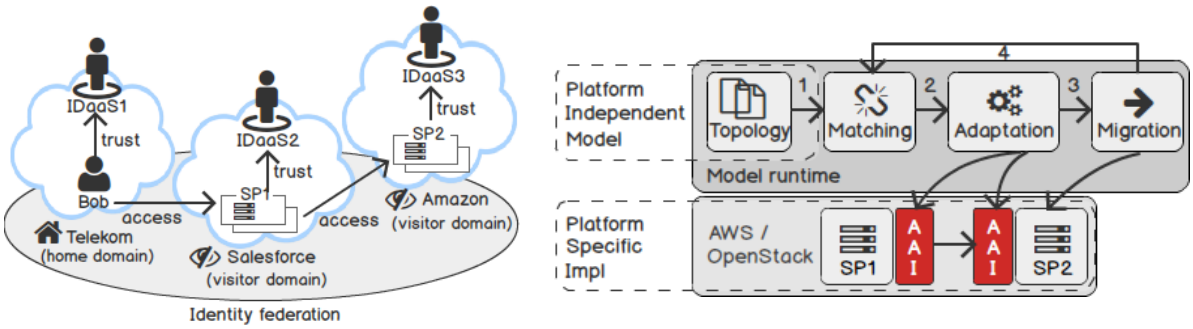
Finally, research in the past has investigated *Purpose-aware Access Control* to prevent one organisation from misusing or accidentally transferring user data to another one [49–52]. A subject may have access to an object if the given *access purpose* corresponds to the *intended purpose* for which data were collected. However, these solutions have difficulties in determining the access purpose of a request. In our research, we add a *purpose authorisation request* that determines an access purpose based on an authentication statement and the requesting service.

4. Architecture Design Overview

In the following sections, we present the design principles for trust adaptation (Section 4.1) and Purpose-based Encryption (Section 4.2).

4.1. Design Principles for Trust Adaptation

We proposed the following trust model for IDaaS [8]. Figure 3a shows three security domains: Telekom, Salesforce, and Amazon. Each domain has an IDaaS. A user Bob registers at his *home IDaaS* (e.g., Telekom) for authentication. The IDaaS in the other domains (e.g., Salesforce) federate with the home IDaaS. We call them a *visitor IDaaS*. In a domain, all SPs trust its IDaaS as an authoritative resource to handle authentication requests (e.g., SP1 trusts IDaaS1, SP2 trusts IDaaS2). This trust model reduces the complexity for a user and an SP to establish trust with each partner individually.



(**a**) (**b**)

Figure 3. (**a**) Trust model; (**b**) Trust adaptation process.

To adapt the security infrastructure for cloud services, our approach considers separation of duties between a *security architect*, an IDaaS, and application developers. We allow the security architect to describe the security infrastructure and express his requirements (the "what"). We let the IDaaS provision the security infrastructure according to the requirements (the "how"). In the end, application developers only need to care about what user attributes are available to them from the environment so they can use them in their applications. To achieve this goal, we also collected the most frequently used security design patterns [33] and modelled the security components.

In Figure 3b, the adaptation process starts by registering the security topologies of cloud services in a modelling engine (i.e., an orchestration engine). In the topology, we describe all trust capabilities (and requirements) of AAI components. When two cloud services sign a B2B contract, the orchestrator matches their trust capabilities and requirements (step 1), then adapts their AAI

implementations in a target cloud hosting (step 2). When a cloud service (e.g., SP2) migrates to another cloud provider (step 3), the adaptation process starts again with input from the new runtime environment (step 1). In general, we do not define a static trust between services, but let the orchestrator establish a dynamic trust between them.

4.2. Design Principles of Purpose-Based Encryption

The GDPR [35] defines lawful regulation for a data controller and a data processor. In particular, the collection of personal data should be lawful under user consent and with a specified purpose. The purpose of data collection has a time limit (from the time to collect data to the time to fulfil the purpose). For later use, PII should not be disclosed for other purposes than the ones they have been collected. Our access control model follows GDPR by taking "time", "purpose", and "domain" as the main factors for describing the disclosure policy. We use the Predicate Encryption with public index [55] to encrypt the data. As a result, each ciphertext is associated with a public index that describes the disclosure policy. According to the "seven laws of identity" [31], we should not limit users to a single authority. Thus, we allow users to encrypt and distribute their data in any security domains. However, only SPs that satisfy the disclosure policy can obtain the key capabilities to decrypt the data. In Figure 4, the Telekom IdP disseminates the encrypted PII on behalf of user Bob to federated domains (step 1). On demand, Bob authenticates to his home IdP, which enables him to access multiple SPs in federated domains (steps 2, 3, and 4).

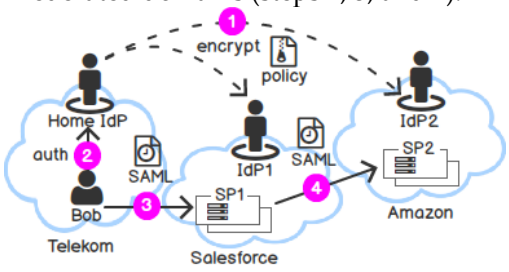


Figure 4. Request flow of Purpose-based Encryption.

5. Trust Adaptation

In the following sections, we dive deeper into the trust adaptation concept. First, we model the security infrastructure. Then, we test the adaptation of the security infrastructure across multiple cloud providers using this model (Section 5.2). Finally, we describe the implementation (Section 5.3) and evaluate the portability and interoperability of the security infrastructure (Section 5.4).

5.1. Topology Modelling

An *application topology* is a description of all application components (e.g., a Web application, a database), their relationships (e.g., the Web application "connects to" the database), and how to deploy these components. The Topology and Orchestration Specification for Cloud Applications (TOSCA) [56] is a standard specification to describe an application topology. In this section, we extend TOSCA to model our security components. In TOSCA, components are defined as *node types*. Relationships between components are defined as *relationship types*. A component may publish some information for other components to establish a relationship with it as *capabilities*.

5.1.1. Node Types

In Figure 5, we model node types (Figure 5a) and capabilities types (Figure 5b) for the security components: "PolicyEnforcementPoint" (PEP), "PolicyDecisionPoint" (not shown in the figure), and "Proxy". Our nodes derive from the normative node type "WebApplication" of TOSCA, thus they have an application endpoint capability by default.

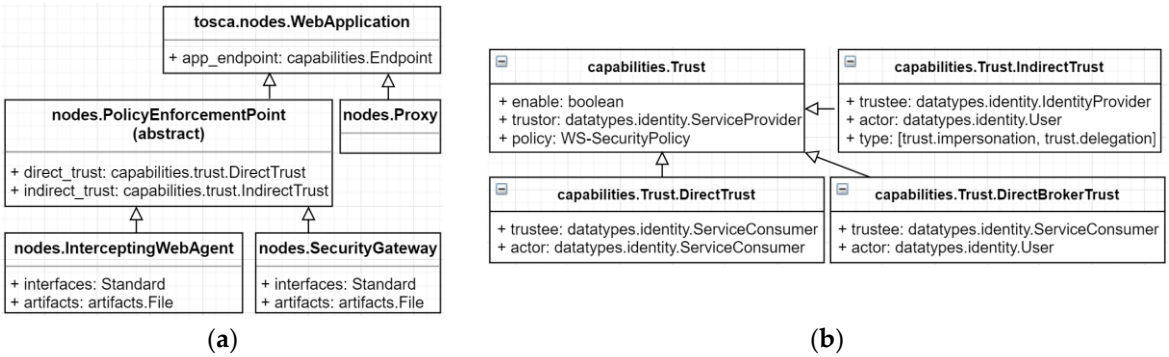


Figure 5. (**a**) Node types; (**b**) Capabilities types.

The PEP node is an abstract node type that does not have any implementation but offers two sub-nodes: an "InterceptingWebAgent" and a "SecurityGateway". A security architect can choose one of these nodes to protect a Web application in the backend. He may choose the "InterceptingWebAgent" that protects the Web application on the same server. Alternatively, he may choose the "SecurityGateway" that enforces centralised protection for multiple Web applications in the backend. Finally, the "Proxy" node is an outbound gateway for a service consumer to connect to an SP. A security architect uses the "Proxy" to intercept outbound requests and to enforce additional security mechanisms to them before calling the PEP of the SP. In TOSCA, all node types have implementation *interfaces* and deployment *artifacts*. The orchestration engine will call the interfaces and use the artifacts to provision the nodes.

5.1.2. Capabilities Types

We model trust capabilities for the security components as follows: From the server's perspective, an SP may define whom it can trust for providing service. In Figure 5b, the "capabilities.Trust" is an abstract trust model. It describes that a *trustor* (i.e., an SP) trusts a *trustee* (to be defined) to delegate permissions for an *actor* subject when the actor subject satisfies the trustor's security policy. If the trust evaluation is correct, the SP continues to execute under the actor's identity. We model trust with three sub capabilities:

1. *DirectTrust*: An SP (trustor) trusts a service consumer (trustee) directly. When using this trust model, the service consumer indicates the SP to execute further using the service consumer's identity itself (actor). This trust model may generate a security policy that builds a mutual trust relationship between Web services such as the "SSL transport binding" [57].
2. *DirectBrokerTrust*: An SP (trustor) trusts a service consumer (trustee) to generate a self-signed identity for a user. The service consumer indicates the SP to execute further using the user identity (actor). This trust model may generate a security policy that builds a mutual trust relationship between Web services and transmits a self-signed user identity such as the "SAML assertion Sender Vouches over SSL" [57].
3. *IndirectTrust*: An SP (trustor) trusts an IdP (trustee) for issuing a user identity. The service consumer may act on behalf of a user (i.e., identity impersonation) or act as the user (i.e., identity delegation) to request a user identity from the IdP. The consumer indicates the SP to execute further using the user identity (actor). This trust model may generate a security policy that establishes trust with an unknown Web service and transmits a user identity such as the "Symmetric binding with supporting token" [57].

In summary, by using this trust model, we can express both direct trust and indirect trust relationships between Web services, and cover all scenarios associated with identity propagation. Also, in our trust capabilities, the trustee and actor have a data type that gives more information about a given entity. For instance, the data type "identity.User" has two elements "claims_mapping" and "group_mapping". These data types allow the security architect to express which identities his application needs from the runtime environment (more details in the experimental results).

5.1.3. Relationship Types

Figure 6a shows how we model a "protect" relationship between an "InterceptingWebAgent" and a "WebService" node, and a "trust" relationship between a Proxy and a PEP. We extend the "protect" and "trust" relationship from the normative relationship "ConnectsTo" of TOSCA. This relationship defines a connection between a source node and a target node. In our case, the "InterceptingWebAgent" is the source node, and the "WebService" is the target node. The relationship also provides standard interfaces such as "pre_configure_source" and "add_target". During the deployment, the orchestration engine calls these interfaces to configure the relationship between them.

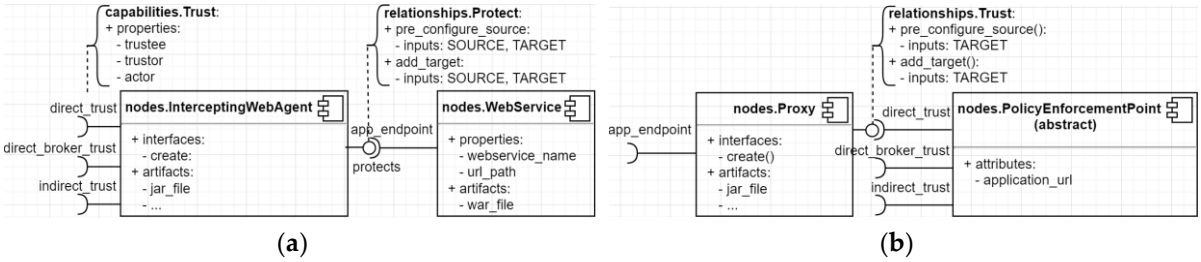


Figure 6. (**a**) Protect relationship; (**b**) Trust relationship.

Figure 6b shows how we model a trust relationship between two Web services. In a service consumer, a security architect can choose if his Proxy shall connect to the PEP (of an SP) via a trust capability. Here, he indicates to which degree the SP obtains a user identity and how they may use it (more details in the experiment results).

5.2. Experimental Results

In the following sections, we prove our concept in a real deployment: Two Web services hosted on two cloud providers establish trust with each other on demand. To show the portability of our concept, the Web services are migrated from a local hosting to a cloud provider and then between two cloud providers. To show the flexibility of our trust model, we consider two scenarios: Web services may belong to the same and to different organisations. First, we describe our test environment (Section 5.2.1). Then, we explain the migration workflow under test (Section 5.2.2).

5.2.1. Test Environment

Figure 7 illustrates two cloud providers: OpenStack [58] and AmazonWS [59]. In each domain, we have set up a Virtual Machine (VM) IDaaS with an orchestration engine and an IdP.

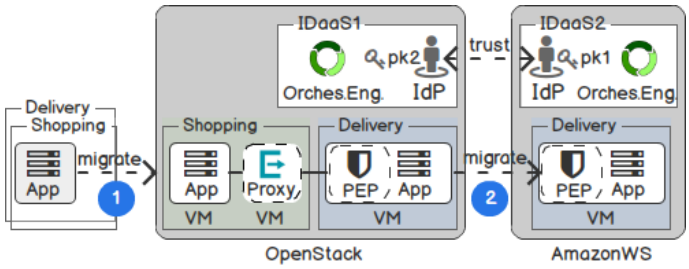


Figure 7. Test environment.

We have developed two Web services to simulate a B2B relationship: A shopping service receives purchase orders from the user and then calls a delivery service for shipping the parcel. In the first scenario, the delivery service is a logistic department from the same company. From the technical point of view, they are two application components belonging to the same tenant, and the delivery service trusts the shopping service for user request authentication. In the second scenario,

the delivery service is an external cloud service from a different tenant. Thus, it needs to authenticate and authorise the user request again.

The migration process happens in the following phases. In the first phase, Web services are developed offline. They are unprotected and migrated to OpenStack. The IDaaS in OpenStack then provisions and updates the AAI implementation for the Web services to trust each other. In the second phase, the delivery service is migrated from OpenStack to AmazonWS. In the second domain, the delivery service not only provides Single Sign-On access for existing users in the IDaaS of AmazonWS but also maintains the existing trust with the shopping service in OpenStack.

We use the open-source software WSO2 [60] and Alien4Cloud [61] to implement the IdP and the orchestrator, respectively. We use Alien4Cloud to control the VMs and the life cycle of the AAI components inside the VMs. We have developed the implementation artifacts and deployment artifacts for these components. At runtime, the orchestrator sends the artifacts to the VM (via SSH) and calls these artifacts during the life cycle of the components. Alternatively, Alien4Cloud can deploy an orchestration agent on the VM (e.g., via Cloud-Init [63]). The orchestration agent then receives our artifacts via a messaging queue in a private network and executes them on the VM [62].

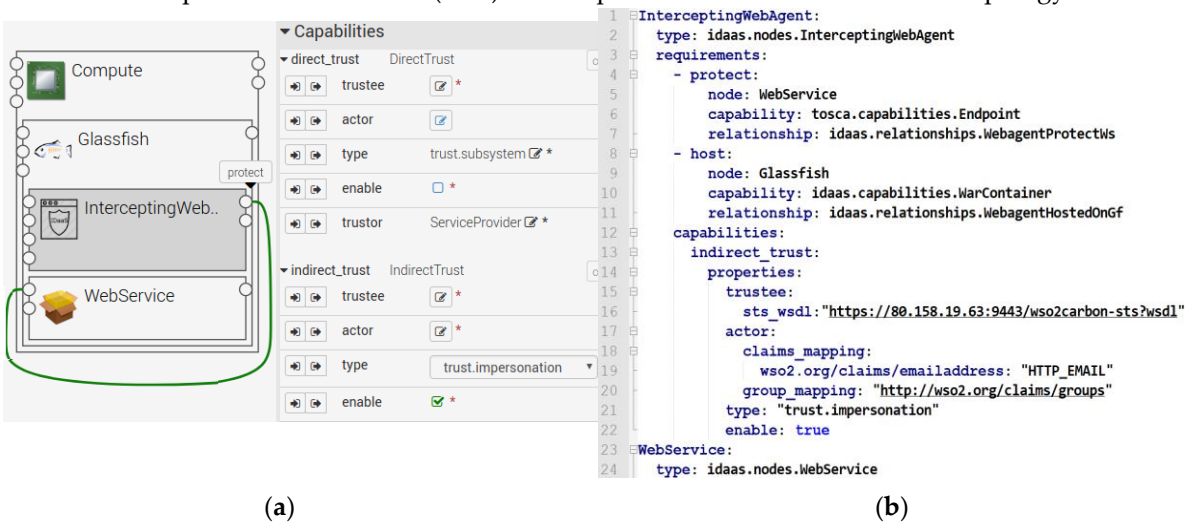
5.2.2. Migration Workflow

(1) Web service development: We demonstrate an example workflow for securing a Java Web service with RBAC. In the development phase, developers have no knowledge of an AAI implementation that an IDaaS offers. However, they may define a set of roles for their application as well as which roles are allowed to access which resources. Figure 8 shows the deployment descriptor of the delivery service, whereby developers allow the role "admin" to access a web resource.

```
<security-constraint>
  ...
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
</security-constraint>
```

Figure 8. Developers may use a deployment descriptor for RBAC.

(2) Topology description: In this step, a security architect describes the security topology. Figure 9a shows a Graphical User Interface (GUI) that helps the architect to describe the topology.



(**a**) (**b**)

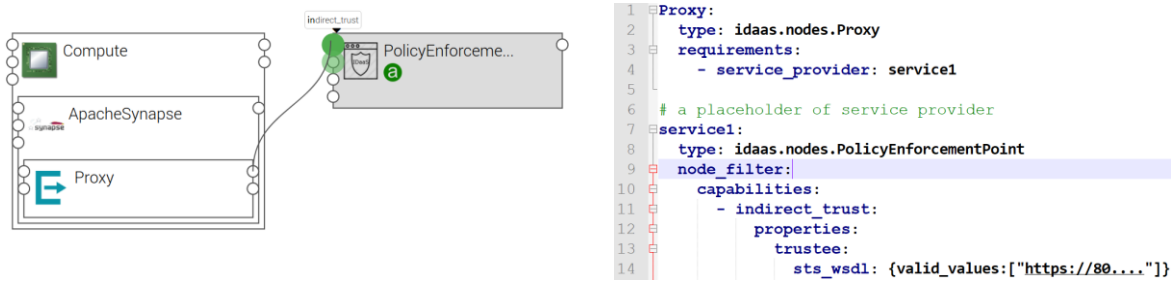
Figure 9. The topology editor (**a**) and topology description (**b**) for the delivery service.

The GUI is the presentation of our topology model. On the left side, the architect defines a "WebService" node running on a "Glassfish" application server of a "Compute" VM. He also wires an "InterceptingWebAgent" node to the application endpoint capability of the "WebService" node to define a "protect" relationship between them. On the right side, he can specify the trust

capabilities of the "InterceptingWebAgent". In the first scenario, the delivery service is from the same organisation, so he chooses the "direct_trust" capability and specifies the public key of the shopping service. In this trust model, the shopping service is the "actor", and the architect can specify a role (e.g., admin) for the shopping service in the "actor" property.

In the second scenario, the service consumer is still unknown, so the architect chooses the "indirect_trust" capability. In this trust model, the user is the "actor", and the delivery service has to authorise the user based on his identity. Here, the architect can specify the endpoint of IDaaS1 as the "trustee" to issue user identity. He may allow the service consumer to act on behalf of the user, so he selects the value "trust.impersonation" in the "type" property (optionally, he may choose "trust.delegation"). When he saves his work, the editor generates an equivalent topology description as in Figure 9b. We can see that the nodes have been instantiated with the trusted endpoint of IDaaS1 (line 16). The architect also specifies that his application needs an email address and a user group from IDaaS1 (lines 18–20) to propagate to the backend.

In the topology for the shopping service (Figure 10a), the architect uses a "Proxy" node to intercept outbound requests from the shopping service and connects the "Proxy" to a "PEP" node.



(**a**) (**b**)

Figure 10. Topology editor (**a**) and topology description (**b**) for the shopping service.

Figure 10b shows the generated topology description for the shopping service. Recall that our PEP node is an abstract node that represents a partner service. The architect may use a TOSCA "node_filter" to define his trust criteria for selecting a partner service (line 9). If the partner service is known beforehand (e.g., the delivery service), he chooses the "direct_trust" capability and specifies the public key of the delivery service. Otherwise, he chooses the "indirect_trust" capability and specifies IDaaS1 as one of the trusted endpoints of the Proxy (line 14).

(3) Provisioning: In this step, an administrator provisions the delivery service to OpenStack. The orchestrator uses the OpenStack APIs (and the credential of the administrator) to boot a VM, deploys the Web service, and calls the interface "pre_configure_source" of the "InterceptingWebAgent" to protect it. For direct trust, the public key of the shopping service is imported to the truststore of the application server. For indirect trust, the Web agent is configured to trust IDaaS1 and intercept client requests on the message layer (see the Web agent implementation).

(4) Integration test of the Web agent: To validate the deployment, the orchestrator calls the interface "add_target" of the "InterceptingWebAgent" to run an integration test. The test sends a dummy SOAP request (with no authentication) to the Web agent endpoint. If the request is denied, the Web service is secured successfully. Otherwise, the orchestrator disables the delivery service from the application server. If the test is successful, the orchestrator publishes the endpoint as well as the trust capability of the delivery service (from Figure 9b) in its registry.

(5) Update: In this step, the shopping service signs a business contract to use the delivery service. An administrator provisions the shopping service to OpenStack. Before the deployment, the orchestrator allows the administrator to confirm the security matching between the two services. Then, the orchestrator sets up the Proxy node according to the trust capability of the delivery service. For direct trust, it imports the public key of the delivery service to the truststore. For indirect trust, it configures the Proxy to trust IDaaS1 and to impersonate the user (see the Proxy implementation).

(6) Integration test of the Proxy: The orchestrator also calls the interface "add_target" of the Proxy node to check if the delivery service's endpoint is accessible. According to the trust capability, the Proxy can call the delivery service directly (i.e., direct trust) or indirectly with the token issued by IDaaS1 (i.e., indirect trust). Otherwise, the test will fail.

(7) Termination: Now the delivery service is migrated from OpenStack to AmazonWS. In OpenStack, the orchestrator removes any credentials of the delivery service from the Proxy node, and then it terminates the delivery service VM.

(8) Migration: In this step, the orchestrator in each domain updates the deployment of the Web service under its control. In the AmazonWS domain, the topology of the delivery service is updated with the new trusted endpoint (i.e., IDaaS2). Then, the orchestrator deploys the delivery service and configures the "InterceptingWebAgent" to trust IDaaS2. In the domain of OpenStack, the delivery service is now an external service. The administrator defines a PEP node (as a substitution for the external service) with the "trustee" updated to the endpoint of IDaaS2. Then, the orchestrator can read this trust capability and configures the "Proxy" node for identity federation (see the Proxy implementation).

5.3. Implementation

We use the open-source software Apache Synapse [64] to implement the Proxy that mediates outbound requests from a Web service client. For direct trust, the Proxy acts as a gateway to sign and encrypt the message on the transport layer. The implementation of direct trust is straightforward by importing the certificate of the Web service to the truststore of Synapse.

For indirect trust, we follow the design pattern of identity proxy in [65]. In this design pattern, the STS Client interacts with two STS and the Web service sequentially from left to right (see Figure 11a). The STS Client first authenticates to STS1 to receive a SAML assertion about a user (step 2) and then submits the SAML assertion to the second one (step 3). The second STS acts as a proxy gateway, which validates the received assertion and transforms it for the STS Client to access the Web service (step 4). In this case, the Web service only trusts STS2 as the proxy gateway for issuing assertions. In Figure 11b, the Web agent receives the SAML assertion from the STS Client (step 1) and propagates the required claims to the Web service (step 2). The Web agent loads the public key of the STS server, which the orchestrator imported previously, and uses it to validate the signature of the SAML assertion. Afterwards, it authenticates the user subject in the SAML to the container-managed authentication.

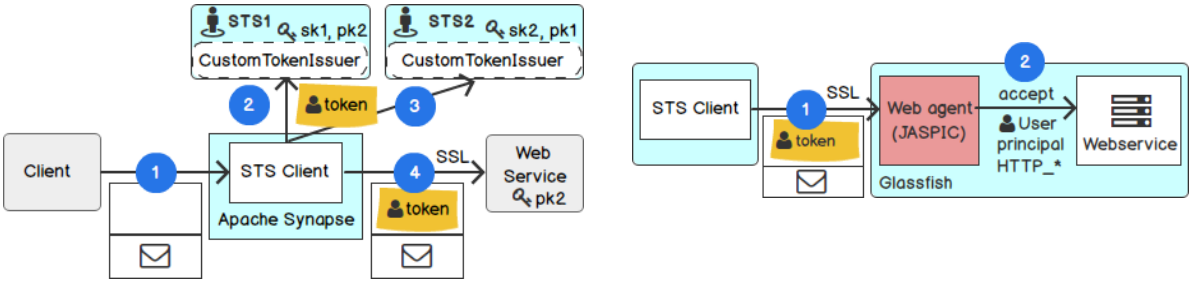


Figure 11. (**a**) Proxy implementation; (**b**) Intercepting Web agent implementation.

5.4. Evaluation

The implementation of the Web agent faces two obstacles of interoperability and portability. Interoperability is the ability of two or more components to exchange information and to use the exchanged information [66]. In our cases, the Web agent needs to exchange the user identity with the application in the backend, but the backend is a black box. Furthermore, developers implement the Web service without knowledge of the AAI implementation at runtime. In our approach, the security architect supports the interoperability between the Web agent and the application in the backend by providing the "group_mapping" and "claims_mapping" in the trust capability. By

reading the "group_mapping", the Web agent understands which claim is mapped to the group principal (e.g., http://wso2.org/claims/groups). By reading the "claims_mapping", the Web agent understands which HTTP headers to propagate to the backend (e.g., HTTP_EMAIL).

It is worth mentioning that, if we implement the Web agent for scripting languages (e.g., PHP or Ruby), we will use HTTP headers, because identity propagation in HTTP headers is considered as programming and platform independent for any applications to obtain information about a request [36]. If we implement the Web agent for Microsoft .NET applications, we will use the Windows Identity Foundation (WIF) [67]. By using WIF, we will not face the obstacles above, because WIF has standardised its identity model for propagating a SAML assertion [68]. In this paper, we evaluate a Java application, because the Web agent implementation in Java has such obstacles for security adaptation to consider.

Portability is the ability to move an application from one cloud provider to another one at the lowest possible cost, effort, and time [66]. The TOSCA specification achieves the portability of cloud applications by using *declarative processing* and *imperative processing* [69]. Declarative processing requires a precise definition of the semantics of node types and relationship types so that any TOSCA compliant orchestration framework can interpret these types and deploy them in a target cloud provider (i.e., the orchestrator has already supported these types and implemented APIs of multiple cloud providers to deploy these types). Because our security nodes inherit from the normative node types of TOSCA (e.g., "WebApplication"), the orchestrator can interpret our nodes as a Web application for the deployment. By inheriting the normative relationship types (e.g., "ConnectsTo" and "HostedOn"), the orchestrator can control the dependency and the life cycle of our components. On the other hand, imperative processing requires explicit instructions on how to manage the components. In our work, we provide deployment artifacts (a Jar archive) and implementation artifacts (shell scripts) for the PEP and the Proxy. These artifacts are examples of imperative processing. Another IDM vendor may replace our artifacts by theirs if they have a specific solution.

6. Purpose-Based Encryption

In the following sections, we dive deeper into the PBE concept. First, we summarise the encryption scheme that we use to encrypt PII (Section 6.1). Second, we show how we use this scheme to encrypt PII (Section 6.2). Third, we present the lifecycle of the encrypted PII in federated domains (Section 6.3). Then, we test the lifecycle of PII with a detailed example (Section 6.4). Finally, we describe the implementation (Section 6.5) as well as evaluate the performance and security considerations of PBE (Section 6.6).

6.1. Multi-Authority ABE Scheme

In this section, we summarise the multi-authority ABE scheme in [55]. In this scheme, multiple authorities can issue secret keys associated with the attributes under their control independently. The scheme consists of the following phases:

(1) AuthSetup (PID, GP) → $\{PK_{PID}, MSK_{PID}\}$: The authority setup algorithm takes a provider identifier *PID* and the global parameters *GP* as inputs. It outputs a public key PK_{PID} and a master secret key MSK_{PID} for the given authority.

(2) KenGen (UID, SESSIONID, MSK_{PID}*, U, GP)* → $SK_{UID,U,SESSIONID}$: The key generation algorithm takes in a user identifier *UID*, a session identifier *SESSIONID*, a master secret key MSK, an attribute *U* (controlled by the authority), and the GP. It outputs a secret key associated with the attribute *U* for the user *UID* in the session identifier *SESSIONID*.

(3) Encrypt (M, P, $\{PK_{PID}\}$*, GP)* → *CT*: The encryption algorithm encrypts a message M with a disclosure policy P, the public key PK_{PID} of the relevant authority, and the GP. It outputs the ciphertext CT.

(4) Decrypt (CT, {SK$_{UID,U,SESSIONID}$}, GP) → *M*: The decryption algorithm takes a ciphertext *CT*, a set of secret keys, and the GP. It outputs the message *M* if the attributes in the secret keys satisfy the disclosure policy *P*. Otherwise the decryption fails.

In the key generation, we use the *UID* concatenating with the *SESSIONID* as an input for tying all key attributes together due to the following reason. The ABE scheme [55] prevents collusion attacks between users (i.e., different users cannot combine their keys to gain more capabilities for decryption). However, the scheme does not prevent various entities from combining secret keys about the same user. This situation happens when an SP in a call chain receives a secret key about a user and may try to collude with other services in a different transaction. In our case, the decryption also fails if SPs try to combine secret keys from various transactions. We call this input a "linchpin" for tying all key attributes of the same user in the same session.

6.2. PII Encryption

Figure 12 shows an example, whereby PII (right column) is encrypted with a disclosure policy (left column). In this example, the data owner "id1" encrypts his data with an intended purpose to complete the "current" transaction (line 3), in particular, "purchase" or "delivery" a product (line 4). The PII is available for SPs managed by the IDM of "salesforce" or "amazon" with the domain "eu" (line 5 and 6). The ciphertext has an "access time" valid within 14 days (line 7 and 8 present a range of dates from 07.09.2018 to 21.09.2018). In the disclosure policy above, we borrow two standardised elements "purpose", "ppurpose" (i.e., primary purpose) from the Platform for Privacy Preferences Project (P3P) [2]: The "purpose" element specifies an intended purpose for disclosing data (e.g., "current", "tailoring", "telemarketing"). If the "purpose" is set to "current", the "ppurpose" provides more details about the current business transaction (e.g., "purchase", "login").

```
1  # INDEX                                  # PAYLOAD (ciphertext)
2  "id1"
3  AND 1 OF ("current")                     <user data>
4  AND 1 OF ("purchase", "delivery")
5  AND 1 OF ("salesforce", "amazon")
6  AND "eu"
7  AND "year2018"
8  AND dayaccess IN (250, 264)
```

Figure 12. An example of PBE.

6.3. The Life Cycle of PII in Federated Security Domains

In the following part, we revisit the scenario where a user Bob from Telekom consumes SPs in Salesforce. Telekom authenticates Bob and issues a *Time Access Token* (TAT). Salesforce is in charge of the purpose authorisation request and issues a *Purpose Access Token* (PAT) for SPs in its domain. An SP can decrypt the PII if it possesses both the time and purpose token. This results in the following request flow as in Figure 13. Briefly, step 2, 4, and 5 represent the standard request flow of Attribute-based Access Control [70]. We only extend the request flow in step 1 and 3.

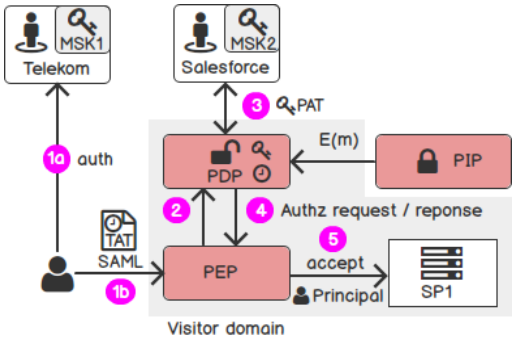


Figure 13. Request flow.

(1) Encryption: Bob encrypts his data with the algorithm in Section 6.1. In particular, he uses the public key of Telekom to encrypt the data with the policies "access time", "domain", and "country". He uses the public key of Salesforce to encrypt the data with the policies "purpose" and "ppurpose". The algorithm outputs one ciphertext for each user attribute and disseminates it to Telekom.

(2) Authentication: Bob accesses SP1 and is redirected to Telekom for authentication. Telekom authenticates Bob and issues a SAML response (step 1a). Telekom uses the key generation algorithm to issue a TAT. This token consists of three key attributes: "access time", "domain", and "country". The key generation also uses the user identifier and the session identifier (in the SAML response) as a "linchpin" for tying all key attributes together. Telekom includes the TAT in a SAML response and returns to the user. In addition, the IdP uses the public key of SP1 to encrypt the TAT so that only SP1 can decrypt it.

(3) Authorisation: Bob gives the SAML response to access SP1 at Salesforce (step 1b). The PEP validates the SAML for authentication (not shown in the figure) and forwards the SAML to the Policy Decision Point (PDP) for authorisation (step 2). The PDP again submits the SAML to the Salesforce's IdP to exchange for a PAT (step 3). Salesforce uses the key generation algorithm to issue a PAT with two key attributes "purpose" and "ppurpose". It also uses the user identifier and the session identifier delivered in the SAML as a "linchpin" for tying all key attributes together.

(4) Decryption: Now the PDP has enough key attributes. It combines the TAT with the PAT. If all key attributes satisfy the disclosure policy and the "linchpin" is the same, the PDP can decrypt the ciphertext. Otherwise, the decryption fails. After the PDP decrypts the PII, it returns the user attributes to the PEP and to the application in the backend (step 4 and 5).

6.4. Experimental Results

In the following sections, we first describe our test environment (Section 6.4.1) and the lifecycle of the PII under test (Section 6.4.2).

6.4.1. Test Environment

As depicted in Figure 14, we have set up a home domain and a visitor domain to simulate Telekom and Salesforce, respectively. In the home domain, we have developed a client application that is responsible for encrypting and storing PII in the home IdP. In the visitor domain, we have developed two Web applications: The shopping service receives purchase orders from the user and then calls the delivery service for shipping the parcel.

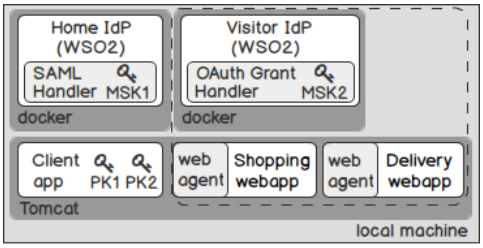


Figure 14. Test environment.

In our proof of concept, we want to show that existing IdP vendors and applications can easily adapt our solution (i.e., existing applications do not need to change their implementations). Therefore, we have developed the following "handlers" that can be integrated into an existing system. In the home IdP, we have developed an OSGi bundle "SAML Handler", which embeds a TAT in an authentication response on the fly. In the visitor IdP, we have developed an OSGi bundle "OAuth Grant Handler", which embeds a PAT in an authorisation response on the fly. We have implemented a Web agent, which intercepts and handles the tokens for the applications. It decrypts the ciphertext and propagates the PII in the payload to the Web application in the backend.

In addition, we have the following key distribution for the components. For encryption, the client application requires the public keys of the home and visitor IdPs (e.g., PK1, PK2). The "SAML

Handler" requires a master secret key of Telekom (e.g., MSK1) for issuing the TAT. The "OAuth Grant Handler" requires a master secret key of Salesforce (e.g., MK2) for issuing the PAT. To isolate the test environment, we test each IdP inside a docker container separately.

6.4.2. The Lifecycle of PII under Test

In the following section, we test the lifecycle of PII. We show that the shopping service and the delivery service can only decrypt a portion of PII to complete a purchase order and to deliver a parcel, respectively, but nothing more.

(1) Encryption: Table 1 shows some examples of user identities under test. In our example, the user is willing to disclose his "date of birth" (DOB) for purchasing a product (e.g., confirm that a user is an adult). However, for delivering a parcel, he does not need to disclose his birthday. Furthermore, he only discloses his "addresses" in a "current" transaction and limited within 14 days after "delivery". In contrast to his "addresses", his "last name" has a longer lifetime (e.g., 30 days) and can be used for contacting purpose without having a "current" transaction (e.g., a marketing service may use his last name to "contact" him later on for advertising a new product).

Table 1. Sample user data.

Attribute Name	Attribute Value	Purpose	PPurpose	Domain	Country Name	Lifetime (days)
Last name	Tri	current, contact	purchase, delivery,	salesforce, amazon	eu	30
DOB	01.01.1980	current	purchase, government	salesforce, amazon	eu	30
Addresses	Berlin…	contact	delivery	salesforce	eu	14

(2) Authentication: Figure 15a shows an authentication response from the home IdP to the shopping service. In the SAML response, the home IdP issues the encrypted TAT (lines 19 to 21) and several attribute statements about the user (lines 12 to 18). These user attributes are in ciphertext format, so the shopping service cannot read them (yet). Figure 15b shows the TAT in plaintext after the services decrypt it. The TAT is a Based64-encoded format that contains some cryptographic key attributes about the current access time (lines 3 to 6), which is the current date that the user authenticates to his home IdP (see the implementation for more details). Also, the TAT has the key attributes issued for the visitor IdP of "salesforce" with the domain "eu" (line 7 and 8).

```
1  <Response Destination="http://.../shopping.com/home.jsp">
2   <Issuer>localhost</saml2:Issuer>
3   <Assertion>
4    <Subject>
5     <NameID>tri</saml2:NameID>
6    </Subject>
7    <AuthnStatement SessionIndex="db0940af-6f7d-485d-..."
8                    AuthnInstant="2017-10-16T14:05:24.415Z"
9                    SessionNotOnOrAfter="2017-10-16T14:20:24.415Z">
10   </AuthnStatement>
11   <AttributeStatement>
12    <Attribute Name="http://wso2.org/claims/dob">
13     <AttributeValue>AAAACAAAAAU...</AttributeValue>
14    </Attribute>
15    <Attribute Name="http://wso2.org/claims/addresses">
16     <AttributeValue>AAAACgAAAAY...</AttributeValue>
17    </Attribute>
18    ...
19    <EncryptedAttribute>
20     <!-- time access token -->
21    </EncryptedAttribute>
22   </AttributeStatement>
23  </Assertion>
24 </Response>
```

(**a**)

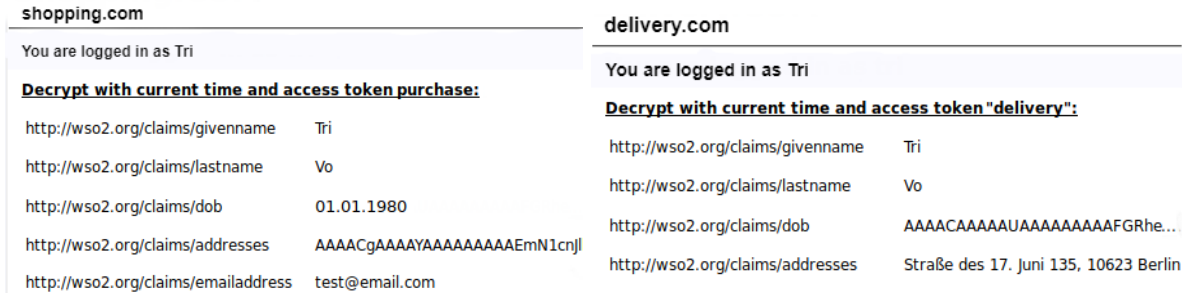
```
1  <Attribute Name="http://wso2.org/claims/timeAccessToken">
2   <AttributeValue>
3    dayaccess549@telekom:Abndy+/AzSNUYM6dZrr30/2kop9/CD...;
4    dayaccess237@telekom:HemLPJ3c4Y4//+jtfKruukl9upnLmU...;
5    ...
6    year2018@telekom:EQlR2W63klkqvJKh8ag2JV46gglsSSaShm...;
7    salesforce@telekom:Ax4xQBous+tU2VCrSSYNZE1SYiL7Q9xK...;
8    eu@telekom:FqqyIaOlFZJstzddoUnYDiNSG3heIzcb32/E/EZ6...;
9   </AttributeValue>
10 </Attribute>
```

(**b**)

Figure 15. (**a**) SAML response with encrypted time access token; (**b**) Time access token.

(3) Purpose authorisation request: To implement the purpose authorisation request, we have extended the protocol "SAML 2.0 Profile for OAuth 2.0" [71]. Briefly, the shopping service authenticates to the visitor IdP and submits the SAML response in a POST request. After receiving the request, "OAuth Handler" checks the expiration time of the SAML session. If the session is still valid, it issues a PAT containing two key attributes "current" and "purchase". It means the shopping service is authorised for processing PII to "purchase" a product in the "current" transaction.

(4) Decryption: In Figure 16a, the shopping service can decrypt PII for the "purchase" purpose (e.g., last name and birthday) but cannot decrypt the "addresses". In Figure 16b, the delivery service also performs the purpose authorisation request and receives a PAT containing the key attributes "delivery", "contact", and "current". With the given keys, the delivery service can decrypt user "addresses" for shipping the product, but cannot see the user birthday.



(**a**) (**b**)

Figure 16. (**a**) The shopping service; (**b**) The delivery service.

(5) Expired authentication: When the shopping service submits an expired SAML session, the PAT does not contain the key attribute "current". Thus, the shopping service cannot decrypt the "dob". It means after the shopping service completed the current transaction, it cannot reuse the SAML to access the "dob" (even though it was authorised to access the "dob" before).

(6) Expired ciphertext: When the user authenticates to the home IdP after 15 days, the home IdP issues a TAT with a new access time. Recall that the "addresses" is encrypted with the disclosure policy limited to 14 days. This time, the delivery service cannot decrypt the ciphertext "addresses". In this scenario, we say that the ciphertext is expired, the IdP has fulfilled the intended purpose for collecting the "addresses" (i.e., to deliver the parcel within 14 days). After 14 days, the user may re-encrypt and disseminate his "addresses", if there is a new intended purpose.

6.5. Implementation

(1) ABE Encryption: We have implemented the ABE scheme [55] by using the jPBC [72], a Java API that wraps the pairing-based cryptosystems written in C [73]. The ABE scheme requires an access policy in the form of a Linear Secret Sharing Scheme (LSSS), but our disclosure policy needs a mixed formula of Boolean expressions, simple thresholds, and numerical range. Therefore, we first convert our policy string to a Boolean formula and then to an LSSS. To convert a numerical range to a Boolean formula, we borrowed the idea of a segment tree [74]. To convert a Boolean formula to an LSSS, we followed the algorithm of Lewko in [75].

(2) Ciphertext serialisation: So far, we have described that we use ABE for encryption. Actually, we do not use ABE to encrypt PII directly but combine it with the Advanced Encryption Standard (AES) [76]. With AES, we can encrypt arbitrarily large data without affecting the performance. In the first step, we generate an ephemeral key and use it as a symmetric key in AES to encrypt the PII. In the second step, we use ABE to encrypt the ephemeral key. For embedding the ciphertext in a SOAP message as well as in an HTTP POST, we encode the ciphertext in a base64 string format.

6.6. Evaluation

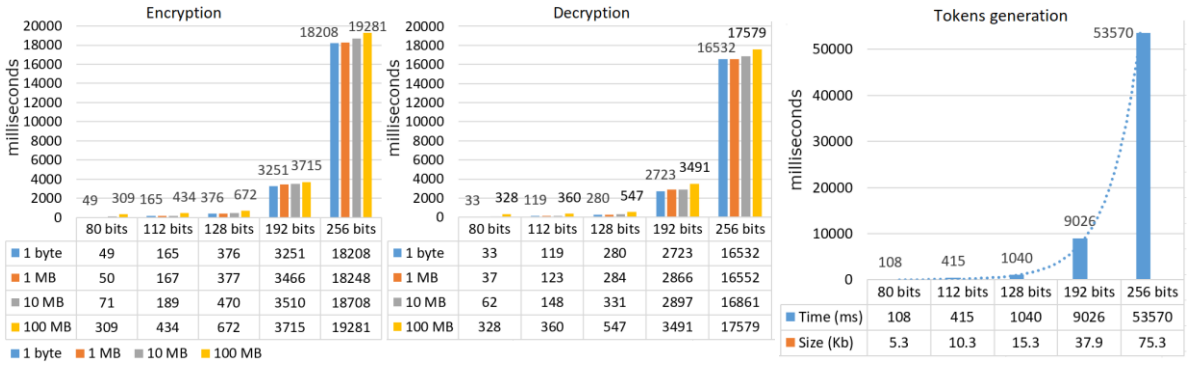
In the following sections, we evaluate the performance of PBE (Section 6.6.1) as well as the benefits and limitations of our approach (Section 6.6.2).

6.6.1. Performance

The performance depends on two factors: the security levels and the numbers of key attributes.

1. *Security levels*: The ABE scheme computes bilinear pairing operations. The National Institute for Standards and Technology (NIST) has recommended the sizes for secure settings of parings and their validity period [77]. Our implementation also follows these key size settings of NIST for both AES and ABE.
2. *Key attributes*: The decryption of the ABE scheme has $(2l + 1)$ pairing operations for l key attributes, and the pairing is an expensive operation. By using the segment tree [74], we can reduce any numerical ranges of size N by a set of size $log_2(N)$ key attributes. Also, we present the access time policy in a "day" unit, which takes at most 10 key attributes. The "day" unit is not as precise as "millisecond". However, most examples of data retention in the P3P [2] are in days, so we think the "day" unit is reasonable.

We have tested the performance with various data sizes (i.e., 1 byte, 1MB, 10MB, and 100MB) and with various security levels (i.e., 80, 112, 128, 192, and 256 bits). Figure 17a shows the performance result on a CPU i7-8650U, 1.90GHz, 16GB RAM. First, we notice that various data sizes in the same security level (i.e., 1 byte, 1MB, 10MB) affect the encryption and decryption time minimally. This is because we use AES to encrypt and decrypt the data. Second, the performance is fast for the following security levels: 80, 112, and 128 bits. To give an illusion, for 128 security bits, it took 377 ms and 284 ms to encrypt and decrypt 1MB, respectively. However, above this level, the performance degrades significantly.



(**a**) (**b**)

Figure 17. (**a**) Performance of encryption, decryption (**b**) Performance of tokens generation on the y-axis in milliseconds.

Figure 17b shows the token generation (TAT and PAT in total) is very fast for 112 and 128 security bits but slow for 192 security bits (9 seconds). Higher security levels also increase the token size slightly. For 112 and 128 security bits, the tokens add additional overhead to the protocol SAML and OAuth (in total) with 10KB and 15KB, respectively. We think this token overhead is acceptable.

In summary, we think the performance is acceptable for 112 and 128 security bits. However, for 192 bits, service consumers will experience a delay in their transactions (around 12 seconds in total). For 256 bits, the performance is not acceptable. To improve the performance, we may implement an ABE scheme that is built on Type F curves [78], which is known to provide faster performance at these levels [79].

6.6.2. Benefits and Limitations

Our approach solves the issues identified in the introduction:

1. *Identity propagation between intermediaries*: Each intermediate service in the call chain (e.g., the shopping and delivery service) uses the SAML response to perform a purpose authorisation request without the user interaction. Therefore, we prevent identity theft in case of users mistakenly decide, which attributes to disclose to a service. According to GDPR, the data controller is accountable for transferring PII to the data processor. That is, the data controller must determine the purposes of PII processing. In other words, when a cloud service registers on a cloud provider, it must provide its business information to the cloud provider such as the access purpose of the service. The IdP use this information to authorise a purpose request.
2. *Honest-but-curious IdP or SP*: Here, we assume that SPs receive the SAML response and the tokens and try to learn more information about PII. First, the home IdP issues the SAML response with its private key. Therefore, SPs cannot modify the SAML response to request the visitor IdP for more capabilities than they are allowed. Second, SPs cannot combine different tokens from different transactions or users to gain more capabilities (i.e., the decryption simply fails if they do so). On the other hand, the visitor IdP can only issue a purpose token for an SP but does not have the time token to decrypt PII. However, our concept cannot prevent the collusion attacks between the visitor IdP and the SPs. Here, we assume that adversaries cannot control both systems at the same time.
3. *Dishonest IdP or SP*: A dishonest entity is an entity that does not follow the protocol. For example, an IdP authorises a wrong purpose (with more capabilities) to an SP or an SP forwards its purpose token to another service. In such cases, an honest entity can prove that the other side has violated the protocol and the dishonest entity must pay for the penalty (according to GDPR, an entity may pay a penalty up to 20 million Euro). In the Facebook incident, Facebook would prove that it did not authorise Cambridge Analytica for the purpose "analysis", so the dishonest application, which forwarded the user data, would pay for the penalty.
4. *Insider attack on an IdP*: PII is encrypted on the IdP. Therefore, an administrator of one IdP does not have enough tokens to decrypt it. However, our approach cannot prevent the collusion attacks between the home and visitor IdP. Here, we assume that adversaries cannot control both IdPs at the same time.
5. *Malicious host*: A malicious host may change the execution of plaintext code to bypass the disclosure policies and access data. In our case, if the host running the shopping (or delivery service) changes the cryptographic computation in anyways, the decryption simply fails.

7. Conclusions and Future Work

In this paper, we proposed two concepts for trust adaptation and privacy-preserving user identity using PBE. In trust adaptation, we moved the trust relationship and identity propagation out of the application implementation and modelled them using TOSCA. We proved that we gain three benefits: First, whenever a cloud service migrates from one cloud provider to another one, the complexity of the security infrastructure does not move with the application infrastructure. We enforced a single security infrastructure that is independent of multiple cloud providers. Second, when the business comes up with a new e-commerce scenario, we proved that IDaaS generates and adapts a platform-specific security infrastructure for the given scenario. Finally, we proved that developers do not need to have specialised security knowledge to understand various protocols, to relearn and implement repetitive tasks using proprietary APIs of any vendors.

We also presented a novel architecture design for privacy-preserving user identity in FIDM. Unlike previous approaches, our novel access control model does not rely on the execution of plaintext code but on the cryptographic computation of time and purpose. We proved that access could be revoked when the ciphertext or the tokens were expired. We showed that existing IDM systems and applications could easily adapt our solution by extending the standard protocols SAML and OAuth. While protecting identity propagation over intermediaries in the backend, the performance is fast for the security levels of 112 and 128 bits. In future work, we may improve the performance of the security levels of 192 and 256 bits.

Our thesis has advanced the field of IDM for cloud computing. However, a number of areas for future work can be built upon the results achieved, especially in edge computing and Internet of Things as follows. In edge computing, sensor devices may need to collect sensitive information from data producer. These devices need to cooperate with each other for some sensitive jobs without the involvement of the users and the data center. When user devices produce data, they automatically encrypt the data with PBE. As a result, only sensor devices from specific domains could process a portion of user data in the given time if they have the right purposes. Notice that we do not define an explicit sensor device that can process the user data in the disclosure policies. Therefore, our concept of PBE protects the confidentiality of user data from unknown devices and from the centralized data storage, of which users may not be aware. If the sensor device or the central data storage is a malicious host and tries to alter the disclosure policies or the tokens, it cannot reconstruct the decryption key (with more capabilities) to access user data.

Author Contributions: director of studies: Woldemar Fuhrmann; conceptualization, methodology, implementation, validation, writing—original draft preparation: Tri H. Vo; supervision, writing—review and editing: Woldemar Fuhrmann, Klaus-Peter Fischer-Hellmann, Steven Furnell.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

Name	Meaning
AAI	Authentication and Authorisation Infrastructure
ABE	Attribute-based Encryption
AES	Advanced Encryption Standard
B2B	Business-to-Business
DOB	Date Of Birth
GDPR	General Data Protection Regulation
IDaaS	Identity-as-a-Service
IDM	Identity Management
IdP	Identity Provider
LSSS	Linear Secret Sharing Scheme
P3P	Platform for Privacy Preferences Project
PAT	Purpose Access Token
PBE	Purpose-based Encryption
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PII	Personal Identifiable Information
SaaS	Software-as-a-Service
SP	Service Provider
TAT	Time Access Token
TOSCA	Topology and Orchestration Specification for Cloud Applications

References

1. ITU-T. *NGN Identity Management Framework, Recommendation Y.2720*; ITU-T. Available online: https://www.itu.int/rec/T-REC-Y.2720/ (accessed on 21 October 2009)
2. Rigo, W.; Matthias, S. The Platform for Privacy Preferences 1.1 (P3P1.1) Specification. Available online: http://www.w3.org/TR/P3P11/ (accessed on 13 November 2006).
3. Chadwick, D. Federated Identity Management. In *Foundations of Security Analysis and Design V SE - 3*; Aldini, A., Barthe, G., Gorrieri, R., Eds.; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2009; Volume 5705, pp. 96–120, doi:10.1007/978-3-642-03829-7_3.

4. Bücker, A.; Werner, F.; Hinton, H.; Hippenstiel, H.P.; Hollin, M.; Neucom, R.; Weeden, S.; Westman, J.; Buecker, A.; Filip, W.; et al. *Federated Identity Management and Web Services Security with IBM Tivoli Security Solutions*; IBM Redbooks, Indianapolis, United States, 2005.
5. Cantor, S.; Kemp, J.; Maler, E. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Available online: http://docs.oasis-open.org/security/saml/v2.0/ (accessed 15 March 2005).
6. Kaler, C.; Mcintosh, M.; Goodner, M.; Nadalin, A. Web Services Federation Language (WS-Federation) Version 1.2. Available online: http://docs.oasis-open.org/wsfed/federation/v1.2/ws-federation.html (accessed on 22 May 2009).
7. Delessy, N.; Fernandez, E.B.; Larrondo-Petrie, M.M. A Pattern Language for Identity Management. In Proceedings of the 2007 International Multi-Conference on Computing in the Global Information Technology (ICCGI'07), Guadeloupe City, Guadeloupe, 4–9 March 2007; pp. 31–31, doi:10.1109/ICCGI.2007.5.
8. Vo, T.H.; Fuhrmann, W.F.; Fischer-Hellmann, K.P. Identity-as-a-Service (IDaaS): A Missing Gap for Moving Enterprise Applications in Inter-Cloud. In Proceedings of the Eleventh International Network Conference, INC 2016, Frankfurt, Germany, 19–21 July 2016; pp. 121–126.
9. Cadwalladr, C.; Graham-Harrison, E. Revealed: 50 million Facebook Profiles Harvested for Cambridge Analytica in Major Data Breach. Available online: https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election (accessed on 17 Mar 2018).
10. Laurent, M.; Bouzefrane, S. *Digital Identity Management*; ISTE Press Ltd: London, UK, 2015.
11. BBC News. Singapore Personal Data Hack Hits 1.5m, Health Authority Says. https://www.bbc.com/news/world-asia-44900507 (accessed on 20 July 2018).
12. Schläger, C.; Sojer, M.; Muschall, B.; Pernul, G. Attribute-Based Authentication and Authorisation Infrastructures for E-Commerce Providers. In Proceedings of the 7th International Conference on E-Commerce and Web Technologies, Krakow, Poland, 5–7 September 2006; Volume 4082, pp. 132–141, doi:10.1007/11823865_14.
13. Andrikopoulos, V.; Binz, T.; Leymann, F.; Strauch, S. How to Adapt Applications for the Cloud Environment. *Computing* **2012**, *95*, 493–535, doi:10.1007/s00607-012-0248-2.
14. Bellendorf, J.; Mann, Z. Cloud Topology and Orchestration Using TOSCA: A Systematic Literature Review. In Proceedings of the 7th IFIP WG 2.14 European Conference, ESOCC 2018, Como, Italy, 12–14 September 2018; pp. 207–215, doi:10.1007/978-3-319-99819-0_16.
15. Cai, Z.; Zhao, L.; Wang, X.; Yang, X.; Qin, J.; Yin, K. A Pattern-Based Code Transformation Approach for Cloud Application Migration. In Proceedings of the 2015 IEEE 8th International Conference on Cloud Computing (CLOUD), New York City, NY, USA, 27 June–2 July 2015; pp. 33–40.
16. Frey, S.; Hasselbring, W.; Schnoor, B. Automatic Conformance Checking for Migrating Software Systems to Cloud Infrastructures and Platforms. *J. Softw. Evol. Process* **2013**, *25*, 1089–1115.
17. Menychtas, A.; Santzaridou, C.; Kousiouris, G.; Varvarigou, T.; Orue-Echevarria, L.; Alonso, J.; Gorronogoitia, J.; Bruneliere, H.; Strauss, O.; Senkova, T.; et al. ARTIST Methodology and Framework: A Novel Approach for the Migration of Legacy Software on the Cloud. In Proceedings of the 2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 23–26 September 2013; pp. 424–431.
18. Kritikos, K.; Massonet, P. Security-Based Adaptation of Multi-Cloud Applications. In *Data Privacy Management, and Security Assurance*; Garcia-Alfaro, J., Navarro-Arribas, G., Aldini, A., Martinelli, F., Suri, N., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 47–64.
19. Lizar, M.; Turner, D. Consent Receipt Specification v.1.1.0. Available online: https://kantarainitiative.org/file-downloads/consent-receipt-specification-v1-1-0/ (accessed on 20 February 2018).
20. Chaum, D. Security without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM* **1985**, *28*, 1030–1044, doi:10.1145/4372.4373.
21. Camenisch, J.; Lysyanskaya, A. An Efficient System for Non-Transferable Anonymous Credentials with Optional Anonymity Revocation. In *Advances in Cryptology—EUROCRYPT 2001*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 2001; Volume 2045, pp. 93–118, doi:10.1007/3-540-44987-6_7.

22. Camenisch, J.; Van Herreweghen, E. Design and Implementation of the Idemix Anonymous Credential System. In Proceedings of the 9th ACM Conference on Computer and Communications Security, Washington, DC, USA, 18–22 November 2002; doi:10.1145/586110.586114.
23. Bendiab, K.; Kolokotronis, N.; Shiaeles, WiP: A Novel Blockchain-Based Trust Model for Cloud Identity Management. In Proceedings of the 16th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC 2018), Athens, Greece, 12–15 August 2018; pp. 724–729.
24. Mikula, T.; Jacobsen, R.H. Identity and Access Management with Blockchain in Electronic Healthcare Records. In Proceedings of the 2018 21st Euromicro Conference on Digital System Design (DSD), Prague, Czech Republic, 29–31 August 2018; pp. 699–706.
25. Bhargav-Spantzel, A.; Squicciarini, A.C.; Xue, R.; Bertino, E. *Practical Identity Theft Prevention Using Aggregated Proof of Knowledge*; CERIAS Tech Report 2006-26; Purdue University: West Lafayette, United States, 2006.
26. Bertino, E.; Lafayette, W.; Paci, F.; Ferrini, R. Privacy-Preserving Digital Identity Management for Cloud Computing. *Identity* **2009**, *32*, 1–7.
27. Guo, N.; Gao, T.; Zhang, B.; Fernando, R.; Bertino, E. Aggregated Privacy-Preserving Identity Verification for Composite Web Services. In Proceedings of the 2011 IEEE International Conference on Web Services, Washington, DC, USA, 4–9 July 2011; pp. 692–693.
28. Chow, S.S.M.; He, Y.-J.; Hui, L.C.K.; Yiu, S.M. SPICE—Simple Privacy-Preserving Identity-Management for Cloud Environment. In *Applied Cryptography and Network Security*; Bao, F., Samarati, P., Zhou, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 526–543, doi:10.1007/978-3-642-31284-7_31.
29. Malina, L.; Hajny, J.; Dzurenda, P.; Zeman, V. Privacy-Preserving Security Solution for Cloud Services. *J. Appl. Res. Technol.* **2015**, *13*, 20–31, doi:10.1016/S1665-6423(15)30002-X.
30. Identity in the Cloud Use Cases Version 1.0. Available online: http://docs.oasis-open.org/id-cloud/IDCloud-usecases/v1.0/cn01/IDCloud-usecases-v1.0-cn01.html (accessed on).
31. Cameron, K. The Laws of Identity. Available online: http://www.identityblog.com/?p=352/#lawsofiden_topic3 (accessed on 13 May 2005).
32. Landau, S.; Moore, T. Economic Tussles in Federated Identity Management. *First Monday* **2012**, *17*, doi:10.5210/fm.v17i10.4254.
33. Vo, T.H.; Fuhrmann, W.; Fischer-Hellmann, K.P. How to Adapt Authentication and Authorization Infrastructure of Applications for the Cloud. In Proceedings of the 2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud), Prague, Czech Republic, 21–23 August 2017; pp. 54–61, doi:10.1109/FiCloud.2017.14.
34. Vo, T.H.; Fuhrmann, W.; Fischer-Hellmann, K.P. Privacy-Preserving User Identity in Identity-as-a-Service. In Proceedings of the 21st Conference on Innovation in Clouds, Internet and Networks, ICIN 2018, Paris, France, 20–22 February 2018; pp. 1–8, doi:10.1109/ICIN.2018.8401613.
35. General Data Protection Regulation, final version, Official Journal of the European Union. Available online: https://eur-lex.europa.eu (accessed on 04 May 2016)
36. Onelogin. Developing with Web Access Management (WAM). Available online: https://developers.onelogin.com/wam (accessed on 17 October 2016).
37. Ping Identity. Available online: https://pingidentity.com (accessed on 10 December 2016).
38. Almorsy, M.; Grundy, J.; Ibrahim, A.S. TOSSMA: A Tenant-Oriented SaaS Security Management Architecture. In Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), Honolulu, Hawaii, United States, 24-29 June 2012; pp. 981–988, doi:10.1109/CLOUD.2012.146.
39. Steel, C. Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management; Prentice Hall, New Jersey, United States, 2005.
40. Vedamuthu, A.S.; Orchard, D.; Hirsch, F.; Hondo, M.; Yendluri, P.; Boubez, T.; Yalçinalp, Ü. Web Services Policy 1.5. Available online: https://www.w3.org/TR/ws-policy/ (accessed on 04 September 2007).
41. Lawrence, K.; Kaler, C. WS-SecurityPolicy 1.2. Available online: http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf (accessed on 25 April 2012).
42. Membrey, P.; Chan, K.C.C.; Ngo, C.; Demchenko, Y.; De Laat, C. Trusted Virtual Infrastructure Bootstrapping for on Demand Services. In Proceedings of the 2012 Seventh International Conference on Availability, Reliability and Security, ARES 2012, Prague, 20-24 August 2012; pp. 350–357.

43. Lang, U. OpenPMF SCaaS: Authorization as a Service for Cloud & SOA Applications. In Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, Indianapolis, IN, USA, 30 November–3 December 2010; pp. 634–643, doi:10.1109/CloudCom.2010.13.
44. Cimato, S.; Damiani, E.; Zavatarelli, F.; Menicocci, R. Towards the Certification of Cloud Services. In Proceedings of the 2013 IEEE Ninth World Congress on Services, Santa Clara, CA, USA, 28 June–3 July 2013; pp. 92–97, doi:10.1109/SERVICES.2013.16.
45. Chadwick, D.W.; Fatema, K. A Privacy Preserving Authorisation System for the Cloud. *J. Comput. Syst. Sci.* **2012**, *78*, 1359–1373, doi:10.1016/j.jcss.2011.12.019.
46. Mont, M.C.; Pearson, S.; Bramhall, P. Towards Accountable Management of Identity and Privacy: Sticky Policies and Enforceable Tracing Services. In Proceedings of the 14th International Workshop on Database and Expert Systems Applications, Prague, Czech Republic, 1–5 September 2003.
47. Beiter, M.; Mont, M.C.; Chen, L.; Pearson, S. End-to-End Policy Based Encryption Techniques for Multi-Party Data Management. *Comput. Stand. Interfaces* **2014**, *36*, 689–703.
48. Ben Othmane, L. *Active Bundles for Protecting Confidentiality of Sensitive Data Throughout Their Lifecycle*; Western Michigan University: Kalamazoo, MI, USA, 2010.
49. Kiernan, J. Hippocratic Databases. In Proceedings of the 28th International Conference on Very Large Data Bases, Hong Kong, China, 20-24 August 2002; pp. 143-154.
50. Byun, J.-W.; Bertino, E.; Li, N. Purpose Based Access Control of Complex Data for Privacy Protection. In Proceedings of the Tenth ACM symposium on Access Control Models and Technologies, SACMAT '05, Stockholm, Sweden, 1–3 June 2005; pp. 102–110.
51. Yang, N.; Barringer, H. A Purpose-Based Access Control Model. *Inf. Assur.* **2007**, *1*, 51–58.
52. Ni, Q.; Bertino, E.; Lobo, J.; Brodie, C.; Karat, C.-M.; Karat, J.; Trombeta, A. Privacy-Aware Role-Based Access Control. *ACM Trans. Inf. Syst. Secur.* **2010**, *13*, 1–31, doi:10.1145/1805974.1805980.
53. Boneh, D.; Franklin, M. Identity-Based Encryption from the Weil Pairing. *SIAM J. Comput.* **2003**, *32*, 586–615, doi:10.1137/S0097539701398521.
54. Sander, T.; Tschudin, C.F. Protecting Mobile Agents Against Malicious Hosts. In *Mobile Agents and Security*; Springer-Verlag: London, UK, 1998; pp. 44–60.
55. Rouselakis, Y.; Waters, B. Efficient Statically-Secure Large-Universe Multi-Authority Attribute-Based Encryption. In *Financial Cryptography and Data Security, Proceedings of the 19th International Conference, FC 2015, San Juan, Puerto Rico, 26-30 January 2015, Revised Selected Papers*; Böhme, R., Okamoto, T., Eds.; Springer: Berlin/Heidelberg, Germany, 2015; pp. 315–332, doi:10.1007/978-3-662-47854-7_19.
56. Rutkowski, M.; Boutier, L. TOSCA Simple Profile in YAML Version 1.1, OASIA Standard. Available online: http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/TOSCA-Simple-Profile-YAML-v1.1.html (accessed on 31 January 2018).
57. Levinson, R.L.; Gullotta, T.; Chang, S.; Raepple, M. WS-SecurityPolicy Examples Version 1.0. Available online: http://docs.oasis-open.org/ws-sx/security-policy/examples/ws-sp-usecases-examples.html (accessed on 4 November 2010).
58. OpenStack. Available online: https://www.openstack.org/ (accessed on 19 March 2018).
59. AmazonWS. Available online: https://aws.amazon.com/ (accessed on 19 March 2018).
60. Gnaniah, S. WSO2 Identity Server Documentation. Available online: http://docs.wso2.com/ (accessed on 19 March 2018).
61. Alien4Cloud version 1.4. Available online: https://alien4cloud.github.io/ (accessed on 19 March 2018).
62. Cloudify Documenation. Available online: https://docs.cloudify.co/ (accessed on 20 May 2016).
63. Cloud-Init Documentation. Available online: http://cloudinit.readthedocs.io (accessed on 17 May 2018).
64. Apache Synapse Enterprise Service Bus (ESB). Available online: http://synapse.apache.org/ (accessed on 19 Apr 2018).
65. Cantor, S. SAML V2.0 Condition for Delegation Restriction. Available online: http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-delegation-cs-01.html (accessed on 31 August 2016).
66. Gonidis, F.; Paraskakis, I.; Kourtesis, D. Addressing the Challenge of Application Portability in Cloud Platforms. In Proceedings of the 7th South East European Doctoral Student Conference (DSC 2012), Thessaloniki, Greece, 24-25 September 2012; pp. 565–576.
67. Microsoft. Windows Identity Foundation. Available online: https://msdn.microsoft.com/en-us/library/ee748484.aspx (accessed on 26 October 2016).

68. Bertocci, V. *Programming Windows Identity Foundation*; Microsoft Press: Redmond, WA, USA, 2011.
69. Leymann, F.; Rutkowski, M.; Hohl, A. Topology and Orchestration Specification for Cloud Applications - Primer Version 1.0, OASIS Committee Note Draft 01. Available online: http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.html (accessed on 31 January 2013).
70. eXtensible Access Control Markup Language (XACML) Version 3.0. Available online: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html (accessed on 22 January 2013).
71. Campbell, B.; Mortimore, C.; Jones, M. Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants. Available online: https://tools.ietf.org/html/rfc7522 (accessed on 31 May 2015).
72. De Caro, A.; Iovino, V. JPBC: Java Pairing Based Cryptography. In Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011, Kerkyra, Corfu, Greece, 28 June–1 July 2011; pp. 850–855.
73. Lynn, B. *On the Implementation of Pairing-Based Cryptosystems*; Stanford University: Stanford, CA, USA, 2007.
74. Attrapadung, N.; Hanaoka, G.; Ogawa, K.; Ohtake, G.; Watanabe, H.; Yamada, S. Attribute-Based Encryption for Range Attributes. In *Security and Cryptography for Networks, Proceedings of the 10th International Conference, SCN 2016, Amalfi, Italy, 31 August–2 September 2016*, Zikas, V., De Prisco, R., Eds.; Springer International Publishing: Cham, Switzerland, 2016; pp. 42–61.
75. Lewko, A.; Waters, B. Decentralizing Attribute-Based Encryption. In *Advances in Cryptology, Proceedings of the EUROCRYPT 2011: 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, 15–19 May 2011*; Paterson, K.G., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; pp. 568–588, doi:10.1007/978-3-642-20465-4_31.
76. Daemen, J.; Rijmen, V. AES Proposal: Rijndael. 1999. Available online: http://www.cryptosoft.de/docs/Rijndael.pdf (accessed on 19 March 2017).
77. Barker, E. Recommendation for Key Management – Part 1: General. In *NIST Spec. Publ. 800-57*; National Institute of Standards and Technology, United States, 2016; pp. 1–142. Available online: http:10.6028/NIST.SP.800-57pt3r1 (accessed on 01.01.2017).
78. Barreto, P.S.L.M. Pairing-Friendly Elliptic Curves of Prime Order. In *Selected Areas in Cryptography*; Preneel, B., Tavares, S., Eds.; Springer: Berlin/Heidelberg, Germany, 2006; pp. 319–331.
79. Galbraith, S.D.; Paterson, K.G.; Smart, N.P. Pairings for Cryptographers. *Discret. Appl. Math.* **2008**, *156*, 3113–3121, doi:10.1016/j.dam.2007.12.010.