frontiers in ICT

ORIGINAL RESEARCH
published: 29 May 2015
doi: 10.3389/fict.2015.00009

SAFAX – an extensible authorization service for cloud environments

*Samuel Paul Kaluvuri, Alexandru Ionut Egner, Jerry den Hartog and Nicola Zannone**

Eindhoven University of Technology, Eindhoven, Netherlands

Cloud storage services have become increasingly popular in recent years. Users are often registered to multiple cloud storage services that suit different needs. However, the *ad hoc* manner in which data sharing between users is implemented lead to issues for these users. For instance, users are required to define different access control policies for each cloud service that they use and are responsible for synchronizing their policies across different cloud providers. Users do not have access to a uniform and expressive method to deal with authorization. Current authorization solutions cannot be applied *as-is*, since they cannot cope with challenges specific to cloud environments. In this paper, we analyze the challenges of data sharing in multi-cloud environments and propose SAFAX, an XACML-based authorization service designed to address these challenges. SAFAX's architecture allows users to deploy their access control policies in a standard format, in a single location, and augment policy evaluation with information from user selectable external trust services. We describe the architecture of SAFAX, a prototype implementation based on this architecture, illustrate the extensibility through external trust services and discuss the benefits of using SAFAX from both the user's and cloud provider's perspectives.

Keywords: access control, cloud, security-as-a-service, XACML, architectural framework

OPEN ACCESS

Edited by:
Luca Viganò,
King's College London, UK

Reviewed by:
Yannick Chevalier,
Université de Toulouse, France
Paolo Mori,
Consiglio Nazionale delle Ricerche, Italy

***Correspondence:**
Nicola Zannone,
Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, Netherlands
n.zannone@tue.nl

Specialty section:
This article was submitted to Computer and Network Security, a section of the journal Frontiers in ICT

Received: *31 January 2015*
Accepted: *30 April 2015*
Published: *29 May 2015*

Citation:
Kaluvuri SP, Egner AI, den Hartog J and Zannone N (2015) SAFAX – an extensible authorization service for cloud environments. Front. ICT 2:9. doi: 10.3389/fict.2015.00009

1. Introduction

Recent years have seen an increased adoption of cloud services, mainly driven by the explosive growth of mobile devices that often rely on cloud services (Gartner Inc, 2014). In particular, the need to have data accessible and synchronized across multiple devices (e.g., smartphones, tablets, PCs) has led to an increased pace of adoption of cloud storage services. In fact, many popular mobile and desktop operating systems already have a tight integration with cloud storage services to allow users to remotely store photos, music, movies, notes, etc. For example, Apple's iPhone is tightly integrated with its cloud storage service iCloud, Android phones with Google Drive, and Windows phones with OneDrive. There are also several other very popular storage services such as Dropbox, SugarSync, and SpiderOak. Given the number and variety of cloud services, end-users are often subscribed to multiple cloud storage services according to their needs.

Besides securely storing their data, users want to be able to share their data with others without compromising on its security. Yet, the use of different cloud providers results in users having several *islands of data*, one for each cloud provider. Thus, the user is faced with the challenging task of managing access rights across several providers.

Current authorization mechanisms implemented by cloud providers are quite primitive, in that users do not have a means of specifying fine-grained access control policies over their data. Furthermore, each cloud provider uses a custom, often *ad hoc*, solution for access control.

This requires end-users to redefine their policies for each cloud storage service, which causes difficulty in synchronizing them across multiple cloud providers. The situation becomes even more challenging when users share their data with other users who are not registered to the same cloud service, as cloud providers do not have any means of verifying authorizations for unknown users.

The successful application of federated identity services in cloud environments (Hühnlein et al., 2010), where services allow their users to log-in based on credentials provided by third-party identity providers such as Google and Facebook, indicates that similar approaches in the context of authorization may be able to resolve the issue of data sharing in a multiple clouds setting. Existing access control standards such as XACML (OASIS, 2013) already provide a baseline for the development of authorization services for cloud environments (Takabi et al., 2010). However, current authorization solutions, even those based on XACML, suffer from severe limitations that hamper their adoption in these cloud environments.

In this paper, we analyze the challenges of sharing data securely in cloud environments and propose SAFAX, a novel XACML-based architectural framework tailored to the development of extensible authorization services for clouds. The key design principle underlying SAFAX is that all components are loosely coupled services, thus providing the flexibility, extensibility, and scalability needed to manage authorizations in cloud environments. SAFAX offers several advantages to both cloud providers and users. It relieves users from the complexity of specifying and maintaining access control policies for each cloud and in different formats. Moreover, users can constrain access to their data on the basis of information that can be retrieved from arbitrary external sources during policy evaluation; SAFAX invokes these services at runtime without any additional configuration by users. From a cloud provider perspective, it helps them to reduce costs associated with implementing custom solutions by outsourcing authorization decision making to an external service.

To demonstrate the feasibility of the proposed framework, we have implemented an authorization service as a web service based on the architectural principles underlying the SAFAX framework. Since the SAFAX authorization service can be used by multiple users (e.g., students, project partners, guest users), we have used the SAFAX authorization service itself as the access control mechanism in order to regulate the actions that users can perform within the SAFAX authorization service. Moreover, we have developed a Graphical User Interface to facilitate **DO**s in the management of policies and in the configuration of the authorization service. We have reported our experience in deploying, using, and securing SAFAX.

The remainder of the paper is structured as follows. The next section introduces a cloud scenario to illustrate the challenges of cross-domain data sharing in cloud environments. Section 3 presents the SAFAX architecture along with its main components, and Section 4 presents a prototype of an authorization service based on SAFAX. Its use in practice is then discussed in Section 5. Finally, Section 6 reviews related work, and Section 7 concludes the paper providing directions for future work.

2. Motivation

In this section, we present a simple scenario that illustrates the challenges of enforcing access control on data stored at different cloud storage services. The scenario focuses on the mechanisms currently used to control the access to data in cloud environments. We first introduce a few terms that are used in the scenario and throughout the paper.

A **Domain** specifies a data set managed by a user, e.g., files stored at a specific cloud storage service.

A **Domain Owner (DO)** is the user to whom the domain belongs.

A **Domains Controller (DC)** is a host of multiple domains, possibly belonging to different **DO**s. A cloud storage service provider is a typical example of **DC**. The **DC** is responsible for guaranteeing the confidentiality, integrity, and availability of the data stored within the domains it hosts.

A **Data Consumer** is a user requesting access to some data.

Alice uses two cloud storage services offered by two independent **DC**s to store her data: *Domains Controller I* (DC_I) and *Domains Controller II* (DC_{II}). She stores personal files, such as music and photos, in her domain AD_1 at DC_I and documents containing sensitive information, such as a business plan, finances, and health records, in her domain AD_2 at DC_{II}. This is illustrated in **Figure 1**.

Alice wants to share her resources stored at domains AD_1 and AD_2 with other users. In particular, she has the following access requirements:

- *AR1*: Bob is allowed to *view* all photos from AD_1;
- *AR2*: Bob is allowed to *view* the business plan from AD_2;
- *AR3*: Any neurologist from Mayo Clinic is allowed to *view* the health records from AD_2 during working hours (09:00 to 18:00);
- *AR4*: Any other action (besides the ones in *AR1*, *AR2*, and *AR3*) for any resource in AD_1 and AD_2 is denied.

At a later point, Alice may need to update these access rights and allow Bob to *modify* the resources as well:

- $\overline{AR1}$: Bob is allowed to *view* and *modify* all photos from AD_1;
- $\overline{AR2}$: Bob is allowed to *view* and *modify* the business plan from AD_2.

Both **DC**s provide Alice (**DO**s, in general) a means of defining access control policies for sharing data with other users. In particular, DC_I enables **DO**s to share resources with other users in three ways: sharing with specific users (usually registered users),

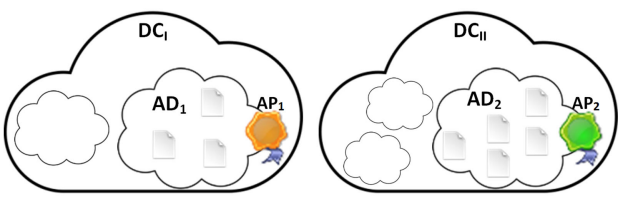

FIGURE 1 | Alice shares resources from multiple cloud providers.

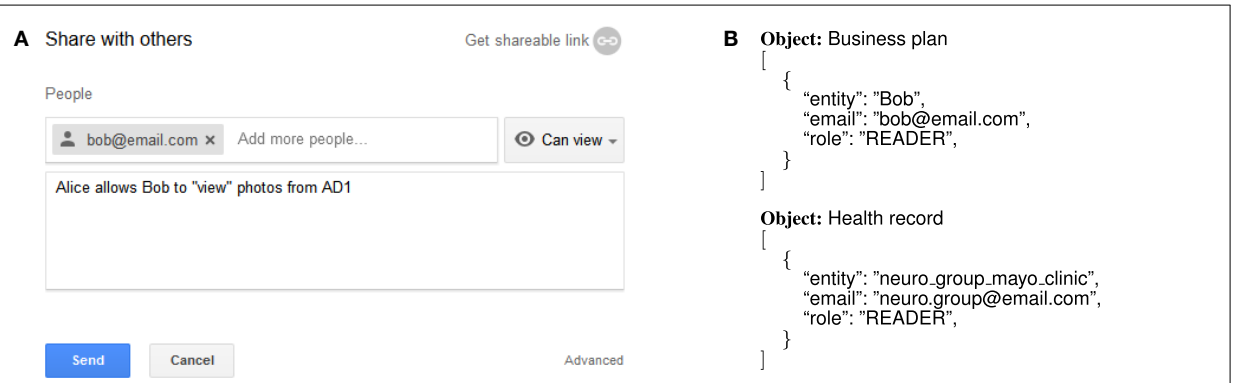

FIGURE 2 | Representation of Alice's access requirements at different cloud providers. (A) Policy AP_1. **(B)** Policy AP_2.

sharing through a link, or making the resource publicly available. This practice is adopted, for instance, by cloud storage services such as Microsoft OneDrive and Google Drive. On the other hand, DC_{II} allows **DO**s to define access rights on buckets and objects using Access Control Lists (ACLs). Through an ACL, **DO**s can define the actions (e.g., read, write) other users (or groups of users) can perform on a certain object. ACLs are currently used by a number of cloud storage services, like Google Cloud Storage and Rackspace Cloud Files, to enable **DO**s to control which data consumer is allowed to access their resources. **Figure 2** shows the policies AP_1 and AP_2 that Alice defines to regulate the access to her data stored in AD_1 and AD_2, respectively. To make the discussion more concrete, we represent these policies using the mechanisms currently offered by existing cloud storage providers. In particular, AP_1 is expressed using the mechanism provided by Google Drive and AP_2 using the mechanism provided by Google Cloud Storage.

One can observe that both AP_1 and AP_2 do not completely reflect Alice's intentions due to the limitations of the authorization mechanisms currently adopted by the cloud providers. Now, we discuss the challenges users have to face when specifying and maintaining policies in a multi-cloud environment.

C1: Identity-based policies. The access control mechanisms adopted by most cloud providers require **DO**s to specify, for each object, the actions that each user can perform on the object, making policy specification costly and error-prone. In addition, **DO**s do not always know the identity of the users, as shown with *AR3*. It is well established that introducing abstract concepts like *role* facilitates the specification and management of policies (Sandhu et al., 1996). Some cloud providers offer some flexibility in this respect by allowing the specification of policies in terms of user groups and folders (next to individual users and objects). However, the properties of users and objects that **DO**s can currently use in the specification of their policies are rather limited. The second rule in **Figure 2B** highlights this problem. Alice can define a group consisting of the neurologists at Mayo Clinic ("*neuro_group_mayo_clinic*") and specify a rule allowing the members of this group to access her health record. However, to define such a group, Alice needs to know all the neurologists at Mayo Clinic and assign them to the group.

C2: Limited expressiveness of policies. Cloud storage providers typically provide **DO**s with a very constrained language for the specification of their policies. For instance, cloud providers using ACLs (such as DC_{II}) only allow the specification of authorizations in terms of user, action, and object. **DO**s are not free to use other context information, such as time and location, to restrict the access to their data. The second rule in **Figure 2B** illustrates that Alice does not have the means to define a specific time interval in which access to her health records is allowed. As a consequence, **DO**s cannot specify fine-grained policies, which are often required to securely share data in dynamic and open systems. Coarse authorization control is recognized by Grobauer et al. (2011) as a main vulnerability of authorization systems currently used by clouds.

C3: Heterogeneity of policy specification. Cloud providers often provide **DO**s with *ad hoc* languages for the specification of their policies. These languages may require the specification of different policy elements to define authorizations. For instance, ACLs are defined in terms of user, action, and object. Other cloud providers do not allow **DO**s to specify the action that users can perform on an object; rather they only consider a general access right that allows users to perform any action on objects. This discrepancy results in **DO**s having to define a different policy to regulate the access to their data for each cloud service provider. For instance, Alice needs to define two different policies, AP_1 and AP_2 for the resources in AD_1 and AD_2, respectively, even if her intention is to assign Bob the same access rights (as exemplified in *AR1* and *AR2*). Moreover, the two providers force Alice to define AP_1 and AP_2 in different formats. DC_I provides Alice a visual interface for specifying AP_1 (see **Figure 2A**), while DC_{II} constrains the specification of AP_2 to a specific language (see **Figure 2B**). This introduces additional burden to **DO**s who have to adapt to a different language and tool for each cloud provider. Therefore, policy specification becomes increasingly complex with the growth of the number of domains, as there is no method for synchronizing policies between different domains (hosted by the same **DC** or by different **DC**s).

C4: No support for cross-domain policy update. The lack of methods for synchronizing policies at different cloud providers can lead to additional issues in policy management. If Alice wants to change the permissions given to Bob (as shown with the updated

rights $\overline{AR1}$ and $\overline{AR2}$), she has to change her policies at each cloud service provider to reflect her new access requirements. This situation could still be acceptable if **DO**s only have to upload the new policy to the different cloud providers. However, as discussed above (challenge **C3**), policies at different cloud providers may have different formats and use different information to make access decisions. This syntactical difference hinders the definition of policies portable between domains.

C5: Lack of cross-domain semantic interoperability. A **DO** specifying a policy to regulate access to its data at different cloud providers expects that every provider interprets and evaluates the policy based on the **DO**s intention. This, however, is difficult to achieve due to syntax differences between the policy languages employed by different cloud providers (challenge **C3**). Standards are often used to smooth these differences and, thus, to enable interoperability in distributed systems. The use of standards, however, does not solve the problem of semantic misalignment of providers' vocabularies (Trivellato et al., 2013). For instance, Google Cloud Storage uses element "role" to indicate the action that a data consumer is allowed to perform (see **Figure 2A**). Other providers may use the element "role" with a different meaning and/or use an element "action" to indicate the action that a certain user can perform. We cannot expect that every provider will employ common data, organizational models, and vocabularies for the specification of security policies.

C6: No empowerment of domain owners. Cloud providers usually evaluate **DO**s' policies relying on information that is locally available (e.g., ACLs). However, this, in combination with the limited expressiveness of the employed policy language (challenge **C2**), provides **DO**s little control over their data. **DO**s should be able to influence the decision of granting or denying the access to their data (this is especially true when dealing with personal information). In particular, **DO**s should be able to define from which source the information used to evaluate their policies is fetched. In *AR3*, Alice wants to constrain the access to her health records to data consumers that are neurologists at Mayo Clinic. Those users, however, may not even be registered with the cloud provider, in which case the properties of the data consumer should be validated by an appropriate trust authority. For instance, in our scenario, the cloud provider should rely on Mayo Clinic to determine whether the data consumer is a neurologist and, thus, authorized to access Alice's health records.

We highlight two additional challenges derived from challenge **C6**, which reflects the perspectives of the data consumer and domains controller.

C7: Required registration of data consumers. Cloud providers usually evaluate and enforce **DO**s' policies based on the data consumer's identity (challenge **C1**). Cloud providers adopt different approaches for handling data sharing with unregistered users. Depending on the provider, **DO**s have the option of sending an invitation to unregistered users, share their data through a link, or even make the data publicly available. The invitation often implies the alteration of the intended policy with additional constraints imposed by the provider. Cloud providers often make the resource *read-only* to an unregistered user, even if the user was given the rights to *modify* the resource. The data consumer, therefore, has to register to the provider to benefit from the intended access rights. Sharing the link has its own drawbacks. Users do not have control over their data as anyone knowing the link can access them. These issues are usually addressed by trust management (Li et al., 2003; Trivellato et al., 2014). However, trust management solutions are typically not integrated in the authorization mechanisms employed by cloud providers. Therefore, the data consumer has to be registered with each cloud provider in order to be granted access to the shared data. In the scenario, Bob is given the right to access specific data from AD_1 and AD_2 (access rights *AR1* and *AR2*), but he needs to have an account at each **DC** in order to access the data.

C8: Limited support for establishing and maintaining a web of trust. From the **DC**s perspective, the problem of unregistered users could be addressed through a *web of trust*, where service providers establish a Federated Identity Management System (Buyya et al., 2010; Chadwick et al., 2011). Within a web of trust, a service provider can trust another provider to issue the credentials proving that an unregistered user satisfies the properties needed to access the data (being a neurologist at Mayo Clinic in our scenario). A risk with this solution, however, is that the web of trust becomes a closed group consisting of only few providers. This risk is aggravated by the burden that maintaining the web of trust with current solutions places on all its members. For instance, each member needs to monitor the trust relationships in the web. For a new service provider to enter a web of trust all existing service providers have to update their components in order to incorporate the new service provider. Moreover, any change in the interfaces of one of the service providers within the web of trust will impact the other service providers.

3. SAFAX – An Authorization Solution for Cloud Services

In this section, we present `SAFAX`, an extensible XACML-based framework suitable for the development of authorization services for cloud. First, we provide a brief overview of XACML and its underlying reference architecture. Then, we present the `SAFAX` architecture along its main components. We also discuss how `SAFAX` meets the challenges presented in the previous section.

3.1. XACML Reference Architecture

As a baseline for the development of an authorization framework that meets the challenges discussed in the previous section, we rely on *Extensible Access Control Markup Language* (XACML) (OASIS, 2013). XACML has become the *de facto* standard for the specification and enforcement of access control policies (Liu et al., 2008); thus, its adoption would facilitate policy interoperability across domains controllers. XACML defines an attribute-based access control policy language implemented in XML as well as a reference architecture for policy evaluation and enforcement. Here, we present an overview of the XACML reference architecture. The XACML policy language is not the focus of this paper, we refer to the XACML Core specification (OASIS, 2013) for its complete description.

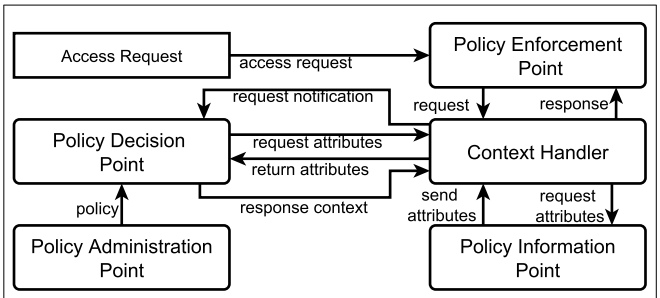

FIGURE 3 | XACML architecture overview.

The XACML reference architecture is shown in **Figure 3**. An access request from an application is sent to the *Policy Enforcement Point* (PEP). The *PEP* forwards the request to the *Context Handler* (CH). If the request is specified in the application's native format, the *CH* constructs a XACML request and sends it to the *Policy Decision Point* (PDP) for evaluation. The *PDP* retrieves the policies from the *Policy Administration Point* (PAP). If additional attributes are required for the evaluation of the access request, the *PDP* queries the *CH* for such attributes. The *CH* retrieves these attributes from the *Policy Information Point* (PIP) and sends them to the *PDP*. The *PDP* evaluates the request against the policies and returns a response specifying the access decision (and possibly a set of obligations to be fulfilled) to the *CH*. The *CH* sends the response to the *PEP*, which is responsible for the enforcement of the decision.

XACML provides a number of extensibility points to customize policy evaluation with respect to the needs of the application domain. The most noteworthy extensibility point is the possibility to augment the PDP with *User Defined Functions* (UDFs) for the specification of custom constraints in the policies. In the next sections, we show how UDFs can be exploited to address (some of) the challenges to be faced by authorization services for multi-cloud environments.

3.2. SAFAX – Overview

SAFAX is an XACML-based architectural framework tailored to the development of authorization services that address the challenges faced by users (**DO**s) while regulating access to their data on cloud storage services. In this section, we present a high level overview of SAFAX and discuss how it overcomes the challenges that were highlighted in Section 2, while the next section provides a detailed view of the main components within the tSAFAX framework.

An overview of the SAFAX framework is shown in **Figure 4A**. The SAFAX framework consists of three main blocks: domain-specific components (i.e., PEP, CH, and PIP); the SAFAX-CORE, which represents the baseline of the authorization service; and trust services, which can be used to evaluate custom constraints in **DO**s' policies. All components forming these blocks are designed as loosely coupled services. SAFAX does not prescribe who should provide these services. In Section 5, we discuss a number of deployment configurations. Here, for description purposes, we assume a configuration in which domain-specific components are under the control of **DC**s, SAFAX-CORE under the control of an authorization service provider and trust services under the control of some providers independent from the authorization service provider. Domain-specific components depend on the application environment. For instance, they should handle the conversion between the attribute representation in the application environment and the attribute representation in the XACML format. The only requirements for domain-specific components within SAFAX are that they adhere to the XACML specification and are offered as services and thus we do not discuss them further.

Two major problems **DO**s have to face when defining their policies are the limited expressiveness of the policy languages currently offered by cloud providers and the heterogeneity of policy specification. SAFAX adopts XACML for policy specification. This standard defines a canonical representation for the inputs and outputs of the PDP (which is the main component of SAFAX-CORE), thus addressing challenge **C3**. In particular, XACML provides an attribute-based policy language in which subjects, actions, and resources are characterized through attributes, thus addressing the inherent limitation of traditional access control models like ACL (**C1**). In addition, the XACML specification provides policy authors a rich set of predefined attributes as well as the possibility to use custom attributes, allowing for the definition of fine-grained policies (**C2**).

The use of XACML addresses some of the challenges for cloud environments and, in particular, the ones concerning policy specification. However, a key aspect is to determine whether a data consumer has the required attributes during policy evaluation. In other words, the relevant attributes of the data consumer requesting the data should be made available to the PDP during policy evaluation. SAFAX anticipates this need and allows **DO**s to constrain the access to their data on the basis of additional trust information that is fetched from external trust services during policy evaluation, for instance, attributes certified by a trusted authority. Intuitively, **DO**s can delegate part of the authorization decision making to trust service providers external to the authorization service provider. The task of connecting external trust sources with SAFAX is hidden from the **DO**s, thus empowering **DO**s with the capability of defining fine-grained access policies and at the same time relieving them from the complexity of maintaining trust sources manually, addressing challenges **C1** and **C6**. In order to achieve this flexibility, we extend the PDP of XACML through UDFs. However, in a major departure from current XACML implementations (Dolski et al., 2007; Liu et al., 2011), we design UDFs as self-configuring clients that consume external trust services. These clients are responsible for generating valid requests for the external trust services and handling the data parsing between the external services and the XACML PDP.

We stress that the authorization service can be outside the control of **DC**s and can be operated instead by an independent provider trusted by **DO**s. Intuitively, the SAFAX framework allows each **DO** to choose an authorization service and to require the **DC**s controlling its domains to consume this service for authorization decision making. This is similar to the authentication mechanism offered by many websites, which allows users to authenticate using the credentials provided by an external identity provider. Thus, SAFAX mitigates the need for **DO**s to define access control policies for each cloud service to which they are subscribed by providing a single point for deploying and maintaining their policies (i.e., a single PAP) and thus addressing challenge **C4**.

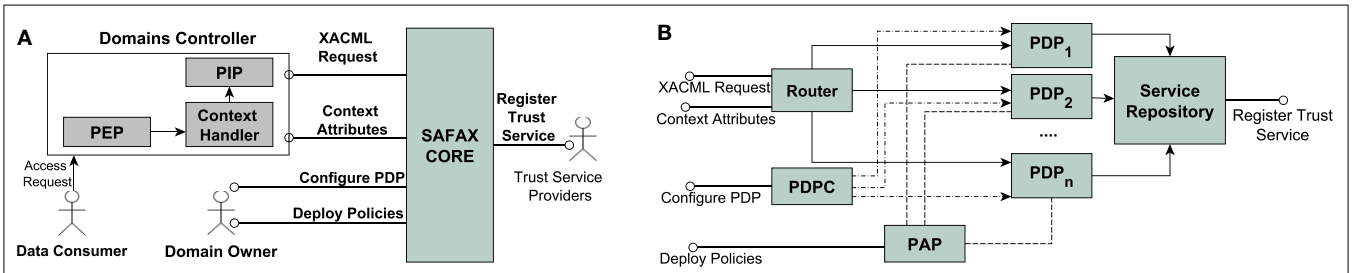

FIGURE 4 | Overview of SAFAX. (A) SAFAX framework. **(B)** SAFAX-CORE architecture.

SAFAX-CORE assigns a dedicated PDP to every **DO**, which is identified with a unique URL (hereafter, we refer to such a URL as PDP-URL). When a **DC** receives an access request for a piece of data, it forwards the request to the authorization service along with the **DO**s PDP-URL. The authorization service uses the PDP-URL to identify the pertinent PDP for evaluation. This implies that the **DC**s should provide a way for the **DO** to specify this PDP-URL as part of their domain configuration settings.

Trust services can be used for a range of purposes such as retrieving trust information from external sources, relocating the computation of complex functions relieving the burden on the PDP and providing additional functionalities. An important example of functionality is enabling policy interoperability in distributed systems. Every **DC** and trust service provider may use a different vocabulary for attribute representation. While XACML unifies policy specification syntactically, it does not solve differences in semantic such as different naming or interpretation of attributes between the entities involved in policy evaluation (**C5**). While existing PDP implementations have no features built in to resolve this issue the extensibility of the SAFAX framework using UDFs allows resolving such issues. By connecting to external services that provides semantic alignment for the relevant attributes without the need of any human intervention (at evaluation time) SAFAX can address challenge **C5**.

The **DC**s, who provide the cloud storage services, can greatly benefit from using SAFAX as a trusted third-party authorization service. In fact, SAFAX reduces **DC**s efforts of creating, maintaining, and monitoring their custom authorization solutions and, thus, they can focus on their core business. **DC**s simply need to consume the SAFAX service. In order to share data with consumers who are not registered with their service, the **DC**s do not have to establish a web of trust with other service providers. Using SAFAX, the **DC**s are completely relieved from this burden and can share data with any data consumer, as long as the properties of the data consumers are captured by the services used for policy evaluation, thus addressing the challenge **C7**. The data consumers also benefit from this approach as they do not have to register to multiple cloud storage providers in order to be able to access data (**C8**).

3.3. SAFAX Architecture

In this section, we focus on the SAFAX-CORE component. SAFAX-CORE exposes several interfaces that can be invoked by different entities interacting with the policy engine. The main interfaces provided by SAFAX-CORE are

- Policy Deployment: allows **DO**s to deploy their access control policies.
- Configure PDP: allows **DO**s to configure their PDPs.
- Register Service: allows the authorization service provider, **DC**s and external trust service providers to register their services with SAFAX.
- XACML Request: handles valid XACML requests from **DC**s.
- Context Attributes: allows **DC**s to augment access requests with additional application-specific attributes.

The components of SAFAX-CORE, as shown in **Figure 4B**, are designed as loosely coupled services thus breaking away from the monolithic component structure often adopted by existing XACML implementations (Dolski et al., 2007; Liu et al., 2011). We provide a brief explanation of the different services that are part of SAFAX-CORE.

Router: This service is responsible for forwarding access requests from **DC**s to the proper PDP based on the **DO**s unique PDP-URL.

Service Repository: This service allows any service registered within SAFAX to discover, bind, and consume other services in a dynamic manner. In addition, this service allows external service providers to register their services with SAFAX. Service registration requires the following details: (a) Service Identifier; (b) Service Provider's Identity; (c) Service Description; (d) Service endpoint (URI); (e) HTTP Method to invoke the service; (f) Request parameters data type; (g) Response parameters data type; (h) Request and Response Messaging format; and (i) Service Type, which indicates whether the service being offered is a PDP service, PAP service, an External Trust Service registered as a UDF, and so on. These details are necessary for services to discover other registered services, configure clients to generate valid requests and parse the responses.

Policy Administration Point (PAP): This service facilitates **DO**s to store and manage their access control policies regardless of the location in which the data to be protected are stored. This service also enables the PDP service to fetch the policies of a specific **DO**. In addition, whenever a policy associated to a **DO** is updated or a new policy is added, the PAP service forces the PDP service to reinitialize the **DO**s PDP in order to reflect the updated policies.

PDP Configuration (PDPC): SAFAX-CORE can support several PDP services; intuitively, every XACML implementation can be seen as a different PDP service in SAFAX. The PDPC service allows **DO**s to select the PDP service to be used for policy evaluation. Moreover, the PDPC allows **DO**s to configure the selected PDP

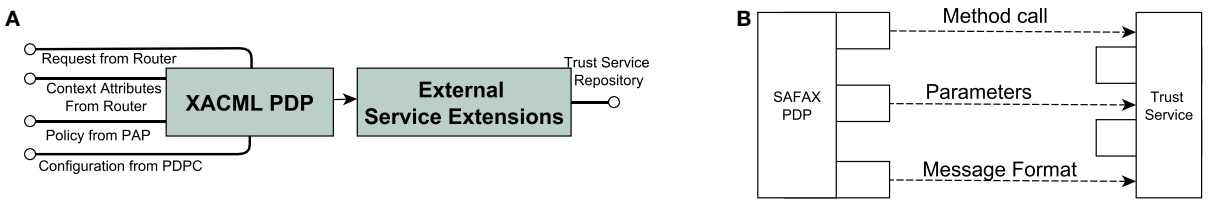

FIGURE 5 | SAFAX PDP and its extensibility with external trust services. **(A)** SAFAX PDP architecture. **(B)** SAFAX plugins.

service by setting a number of parameters including the root combining algorithms[1]. These configuration settings can be changed at any time and pushed to the PDP service, which in turn initializes (or reinitializes) the **DO**s PDP to reflect the changes.

Policy Decision Point (PDP): SAFAX assigns a dedicated PDP to each **DO**. Every PDP is identified by a unique URL (PDP-URL). The PDP assigned to a **DO** handles all the access requests for its data regardless of the **DC** which sends them. The PDP fetches the policies associated to the **DO** from the PAP service. In addition, this service allows the PDPC service to push the PDP configuration settings specified by the **DO**. In SAFAX, we decouple UDFs from the XACML PDP component to allow the framework to be extensible without disrupting existing components. As shown in **Figure 5A**, the PDP consists of a XACML PDP and the *External Service Extensions* component that handles the invocations to external trust service. The *External Service Extensions* component contains, for each UDF, a self-configuring client that consumes the external trust service corresponding to the UDF. These clients are generic and self-configure them selves based on the information related to a given UDF that has been registered by a trust service provider with the SAFAX-CORE service. They read the external trust service description from the Service Repository and, accordingly, build a valid request and communicate with the service using the specified protocol. Moreover, they parse the response from external trust services to retrieve the trust information requested for policy evaluation.

External Trust Services: We decouple UDFs from the PDP and implement them as external, but pluggable services, as shown in **Figure 5B**. This pluggable architecture allows SAFAX to be *extensible* in order to serve access requests that require complex processing in a *scalable* manner as well as consuming information from external sources. This way, the processing of trust information can be outsourced to external services while SAFAX acts as an orchestrator for these service calls based on the **DO**s policies. External service providers that want to plug-in their services with SAFAX need to comply with two main constraints: (i) register their service through the Service Repository of SAFAX along with a UDF Identifier which will be used by the **DO**s in their access control policies; and (ii) external service APIs must conform to implementation dependent specifications of SAFAX. As shown in **Figure 5B**, each provider that registers their service as an external trust service in the SAFAX framework must specify: (a) the *Method call* through which the service can be invoked; (b) the parameters that must be supplied to the service; and (c) the message format that indicates how the parameters will be packaged.

3.4. Message Flow within SAFAX

The services presented in the previous section give an indication of the functionality and capabilities of each component. In this section, we present the overall integrated view of the SAFAX architecture and its usage.

DOs specify access control policies for their data, which can be stored across multiple cloud providers (**DC**s). For instance, in the scenario given in Section 2, Alice wants to restrict the access to her health records to neurologists at Mayo Clinic. This access requirement can be represented using the XACML rule in **Figure 6**. Lines 35–42 show an example of UDF: function `urn:nl:tue:sec:pdp:1.0:udf:credential:user:has:credential`. This UDF takes as input a user name, a credential, and an issuer, and returns a Boolean value indicating whether the user has the specified credential from the specified issuer. It is evaluated using an external credential-based trust service (see Section 4) during policy evaluation.

DOs use the PAP to deploy the specified policies and the PDPC to configure their PDP by selecting a default root combining algorithm, the PDP service to be used for policy evaluation, etc. SAFAX provides the **DO** a unique PDP-URL to invoke the assigned PDP. All access requests for data belonging to the **DO** are forwarded to the assigned PDP through the unique PDP-URL. As a consequence, **DO**s should be able to configure the PDP-URL at every **DC** they are subscribed to.

Moreover, each **DC** should implement the domain-specific components (i.e., PEP, CH, and PIP) and register the Context Handler service with the SAFAX framework. This is necessary since during policy evaluation the PDP might need additional attributes from the CH of the **DC** such as the IP addresses of the data consumer, system time, and other attributes that can be required for policy evaluation. In particular, the CH interface to fetch the attributes should be registered with the service registry of the SAFAX framework.

Figure 7 shows the interaction of services within SAFAX. We describe the message flow by following a concrete scenario that takes place in the context presented in Section 2: Charlie (a neurologist from Mayo Clinic) wants to *view* the health records that Alice stored on a specific domain (AD_2) of the domains controller DC_{II}.

Whenever a data consumer (Charlie) sends a request to access data belonging to a **DO** (Alice) to a **DC** (in our scenario to DC_{II}), the access request is intercepted by the PEP of the **DC**. The PEP forwards the request to the CH service, which first fetches additional information from its PIP, enriching the original

[1]A root combining algorithm is needed to resolve policy conflicts that can arise when more than one policy is deployed in the PDP (OASIS, 2013).

```
<Rule Effect="Permit" RuleId="urn:nl:tue:sec:example:sample:alice">
  <Target>
    <Resources>
      <Resource>
        <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            healthRecord
          </AttributeValue>
          <ResourceAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            read
          </AttributeValue>
          <ActionAttributeDesignator
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
            DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Condition>
    <Apply FunctionId="urn:nl:tue:sec:pdp:1.0:udf:credential:user:has:credential">
      <SubjectAttributeDesignator
        AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        neurologist
      </AttributeValue>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        MayoClinic
      </AttributeValue>
    </Apply>
  </Condition>
</Rule>
```

FIGURE 6 | XACML rule encoding Alice's authorization requirement *AR*3.

request, and then constructs a valid XACML request. The CH service forwards the XACML request to the PDP assigned to the **DO** by invoking the SAFAX service. The Router intercepts the access request coming from the CH and forwards it to the proper PDP based on the PDP-URL. During policy evaluation, if the PDP needs additional attributes (e.g., timestamp), it contacts the **DC**'s Context Handler service. When the **DO** specifies policies that require additional (external) trust information to make a decision, the PDP service contacts the external trust services. In our scenario, the PDP contacts Mayo Clinic, which can been seen as the trust service provider for the trust information specified by Alice in her policy (i.e., the issuer of the credential), to determine whether Charlie is a neurologist at the Clinic. After receiving the required information, the PDP computes the decision and informs the **DC**s Context Handler. The CH parses the XACML response and sends the decision to the PEP. The PEP enforces the decision by allowing or denying access to the **DO**s data to the consumer based on the given decision.

4. Implementation

We have realized the SAFAX architecture by implementing an authorization service as a web service following the architectural principles and guidelines given in Section 3[2]. The SAFAX authorization service is implemented in Java running on an Apache Tomcat server and using Jersey as the service framework. MySQL is used as a back-end persistent data storage.

The authorization service exposes interfaces that can be invoked by the PEP implemented by **DC**s. Moreover, for the sake of completeness and alignment with the XACML reference architecture, we have implemented examples of PEP, PIP, and CH services.

In order to ease the management of policies and configuration of PDPs by **DO**s, we have developed a User Interface (referred as SAFAX GUI) that communicates with the SAFAX services. SAFAX GUI is developed using HTML, CSS, and AJAX to consume SAFAX services. The SAFAX GUI enables a **DO** to create a *project*, which can contain multiple *demos*. Intuitively, a demo corresponds to the management point for regulating access to the data in a **Domain**. Each demo is associated to a single PDP, and **DO**s can upload their access control policies to the PAP service and configure the PDP through *Demo Management* interface as shown in **Figure 8**. **Figure 8A** shows the part of the SAFAX GUI that allows **DO**s to deploy their access control policies. For demonstration purposes, we facilitate **DO**s to upload sample access requests to test their policies. **Figure 8B** shows how xtbfDOs can configure their PDPs as well as providing them the URL through which their PDP can be invoked. **Figure 8C** shows how the sample PIP can be configured by uploading attributes that can be retrieved during policy evaluation.

[2]The authorization service is available at http://security1.win.tue.nl/safax/

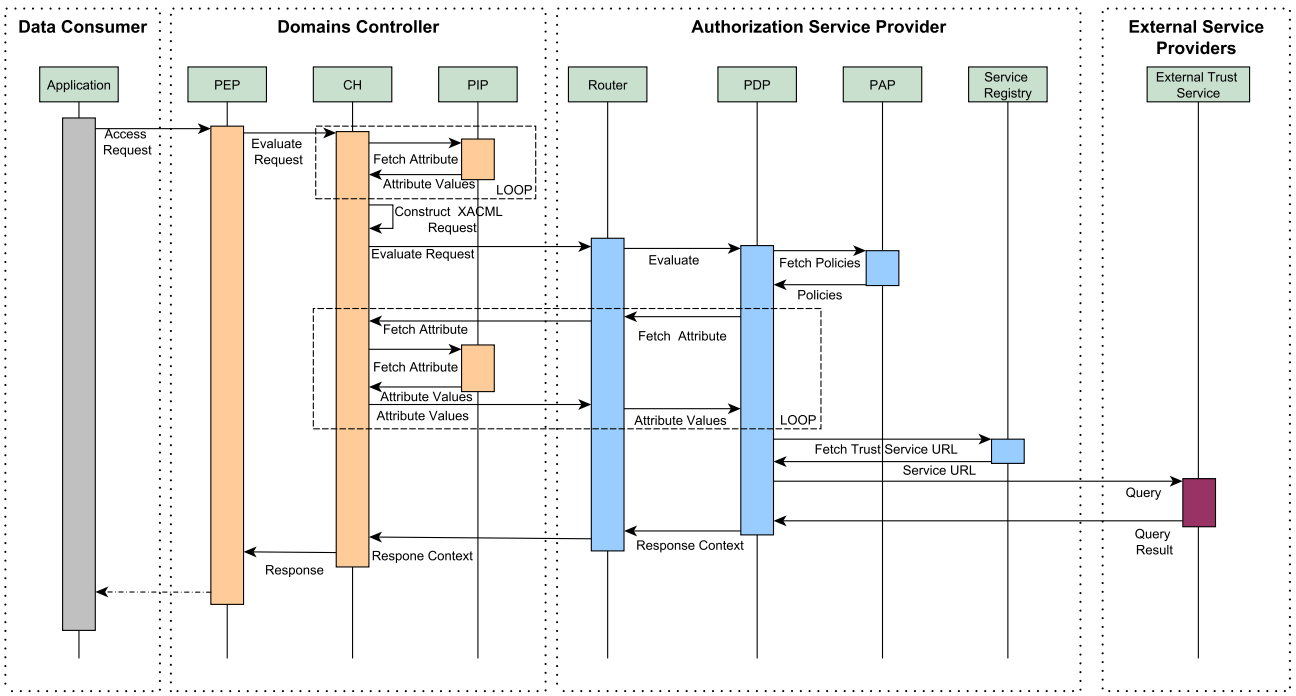

FIGURE 7 | Message flow for policy evaluation using SAFAX.

We have also built a service that simulates multiple **DC**s by allowing the **DC**s to evaluate their policies against different request. This service is accessible through the SAFAX GUI, which makes it possible to test the SAFAX authorization service in various scenarios. A screenshot of the GUI for policy evaluation is shown in **Figure 9A**. In addition to that, we provide an additional feature through which **DO**s can analyze the log generated during the evaluation of access requests to their PDP (**Figure 9B**). This is useful for **DO**s to gain feedback regarding their policies and correct them if necessary.

We have chosen to implement SAFAX services as RESTful services that strictly conform to the following constraints:

- Services are purely stateless and all information related to the state is contained either with the request or stored in a persistent database.
- Services communicate with each other either in JSON or XML.

The *Service Repository* component uses a MySQL back-end database to store the information regarding a service. Every PDP that is assigned to a **DO** is treated as an independent service and is registered with the service repository.

The *Router* parses the PDP-URL, which is used by the **DC**s to invoke the PDP assigned to a **DO**, extracts the code used to identify the PDP object within the SAFAX framework, and forwards the access request to that particular PDP object. For each access request, the Router generates a unique *Transaction ID*, internal to the SAFAX framework and forwards it to the PDP along with the access request. This is used for correlating the request-responses with the PDPs, and identifying the log messages generated by the PDP during the evaluation of a particular access request (**Figure 9B**).

Our implementation of the PAP uses a MySQL database to store the uploaded policies and make them available to the PDP. In addition to XACML policies, for demonstration purposes, we allow **DO**s to upload policies for external services that will be deployed on the respective external trust services. **DO**s can use the SAFAX GUI to upload their policies and deploy them on their assigned PDPs and trust services.

In our prototype, we used HERAS-AF (Dolski et al., 2007) as the default XACML PDP implementation. HERAS-AF was chosen since it is an open-source project, actively developed and well supported. HERAS-AF PDP component is provided as a JAR file, which we wrap as a service by exposing the interfaces for policy deployment and access requests as web interfaces. HERAS-AF supports UDFs, but they have to be bound to the PDP implementation. To circumvent this limitation, we have extended the core HERAS-AF with several classes that: (a) read the **DO**s configurations from the PDPC and initialize the PDP accordingly; (b) communicate with the Service Repository to fetch the configurations of external trust services based on the policies deployed by the **DO**; and (c) invoke external trust services and parse the response (represented in JSON) to retrieve the values from those services.

Although HERAS-AF has been used in the current implementation of SAFAX, this is not a constraint for the SAFAX architecture. In fact, the extensions to HERAS-AF are made in a manner that replacing HERAS-AF with any other XACML implementation would require changing a few lines of code in our implementation.

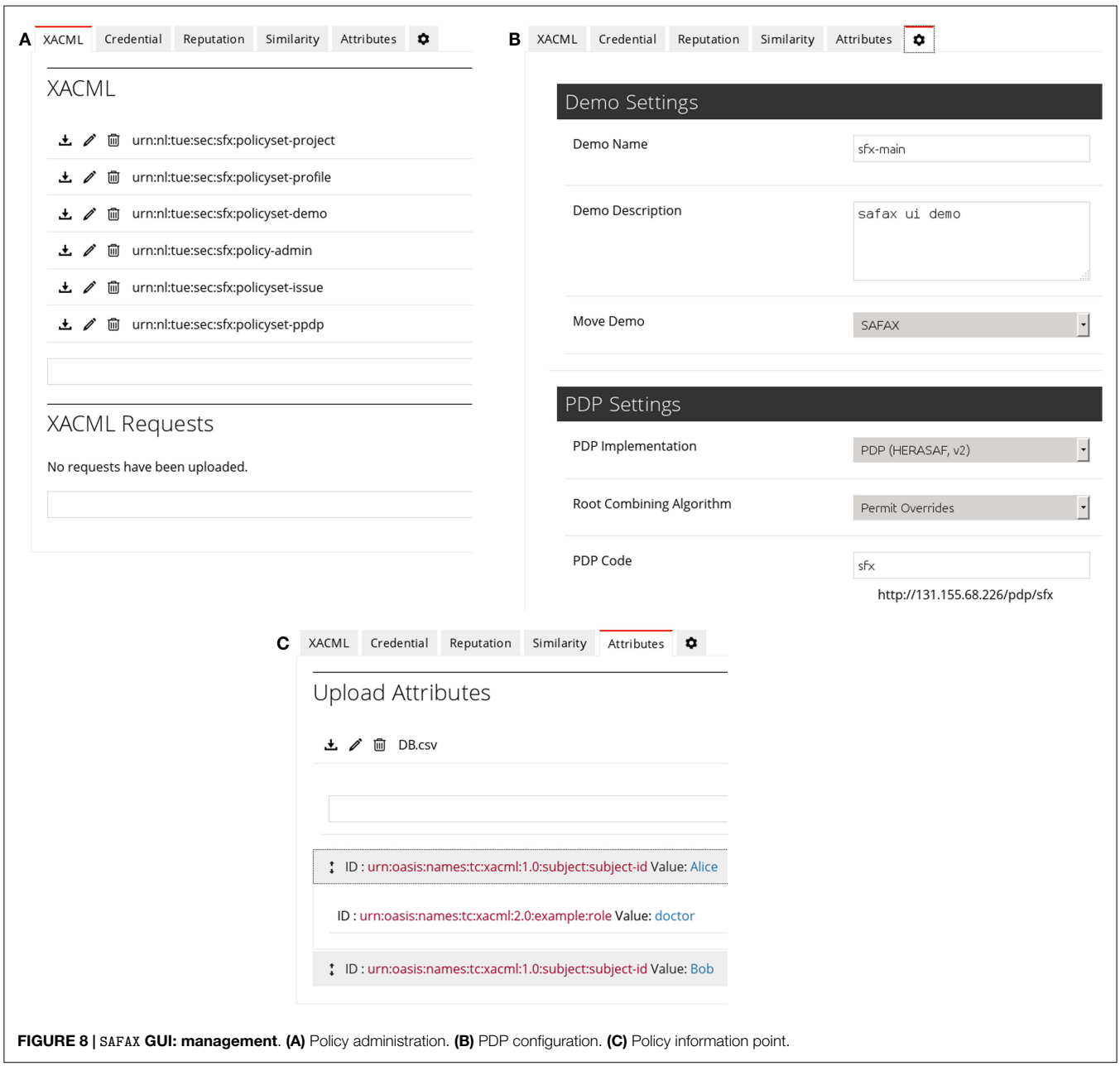

FIGURE 8 | SAFAX GUI: management. (A) Policy administration. **(B)** PDP configuration. **(C)** Policy information point.

Our prototype implementation of SAFAX currently supports four external trust services: a credential-based trust management service, two reputation services and a policy alignment service. The credential-based trust management service is based on GEM (Trivellato et al., 2014), a distributed goal evaluation algorithm for trust management systems. One reputation service uses a flow-based reputation metric (Simone et al., 2012), which extends EigenTrust (Kamvar et al., 2003) by providing absolute reputations values. The other reputation service uses a subjective logic system centered on evidence (Skoric et al., 2014), thus providing reputation values along with uncertainty. Finally, the policy alignment service is an ontology-based service that uses semantic alignment techniques to map concepts from different ontologies (Trivellato et al., 2009), allowing domain owners to use different vocabularies and models for the specification of their policies. These services communicate with the SAFAX PDP using the RESTful protocol. This variety of trust services allows **DO**s to use different types of trust information within the evaluation of their policies, illustrating the flexibility of SAFAX. Note that the aforementioned trust services are simply provided as an example of how SAFAX can be extended to outsource the processing of trust information to external services. A full description of these services is out of the scope of this paper.

5. Discussion

In this section, we present various trust models that are typically employed in cloud environments and discuss their impact

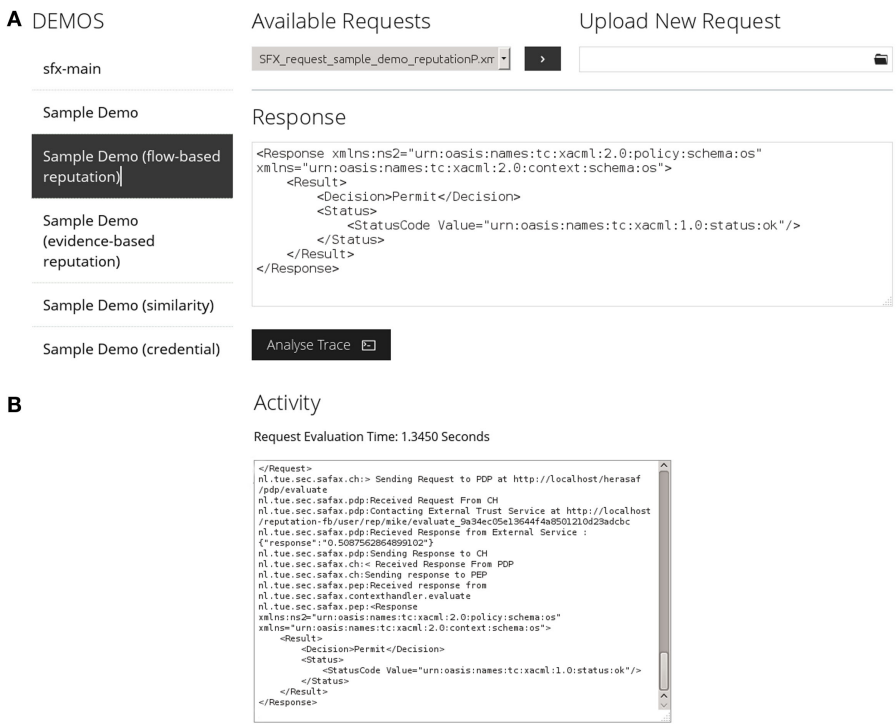

FIGURE 9 | `SAFAX` **GUI: evaluation**. **(A)** Policy evaluation. **(B)** Evaluation log.

on deployment configurations of `SAFAX`. We also discuss our experience of deploying, using, and securing `SAFAX`.

5.1. Trust Models in Cloud Environments

Trust models in cloud environments have a direct impact on the deployment configurations of authorization solutions based on the trust relationships and the control that entities have on the various services. We can broadly classify the trust models into three categories: (a) Centralized Trust; (b) Distributed Trust; and (c) Distributed and Delegated Trust. A representation of these trust models is provided in **Figure 10**.

5.1.1. Centralized Trust Model

The **DC** has control over all the services forming the authorization service including the trust services (**Figure 10A**). In particular, the **DC** deploys the domain-specific components (PEP, CH, and PIP) as well as the `SAFAX-CORE` components and trust services within its organizational perimeter. However, trust services can only consume information that is internal to the **DC** or can be used to extend the functionality of the XACML PDP. The **DO** trusts the **DC** for the secure storage of their access control policies, for the correct evaluation of access requests from data consumers and ultimately for the enforcement of their policies. This trust model is suitable for very sensitive domains such as military clouds where the storage and evaluation of access control policies cannot be outsourced to an external entity. From the **DO**s perspective, this model has, however, most of the limitations discussed in Section 2 as the **DO** does not have the capability to freely specify its policies nor to decide which trust services should be used for policy evaluation. These aspects remain under the control of the **DC**.

5.1.2. Distributed Trust Model

In this model, the **DC** is not responsible for the evaluation of **DO**s' policies (**Figure 10B**). This task is outsourced to a dedicated authorization service, possibly selected by **DO**s. In particular, the authorization service provider deploys the `SAFAX-CORE` components within its organizational perimeter. Trust services that complement the authorization service are also provided by the same authorization provider and thus they are in the same domain. In this trust model, trust services are mainly used to extend the functionality of the PDP; for example, a trust service can process an IP address to determine the country to which the IP address belongs or provide support for spatial data types and spatial authorization decision functions (Matheus, 2012). In essence, trust services cannot be used as external sources of information. The **DC** only deploys the domain-specific components of `SAFAX` (PEP, CH, and PIP) for the management and enforcement of authorization decisions within its organizational perimeter and consumes the authorization service through the CH component. As shown in **Figure 10B**, the **DO** trusts the authorization service for the secure storage and evaluation of its access policies and trusts the **DC** to consume the authorization service for the evaluation of access requests and to enforce the authorization decision. This trust model is suited when the **DO** is subscribed to multiple cloud services and needs a unified means to specify and maintain its access control policies. This model is ideal for **DC**s that want to externalize the policy evaluation.

5.1.3. Distributed and Delegated Trust Model

As in the distributed trust model, the **DC** consumes the authorization service while implementing the domain-specific components

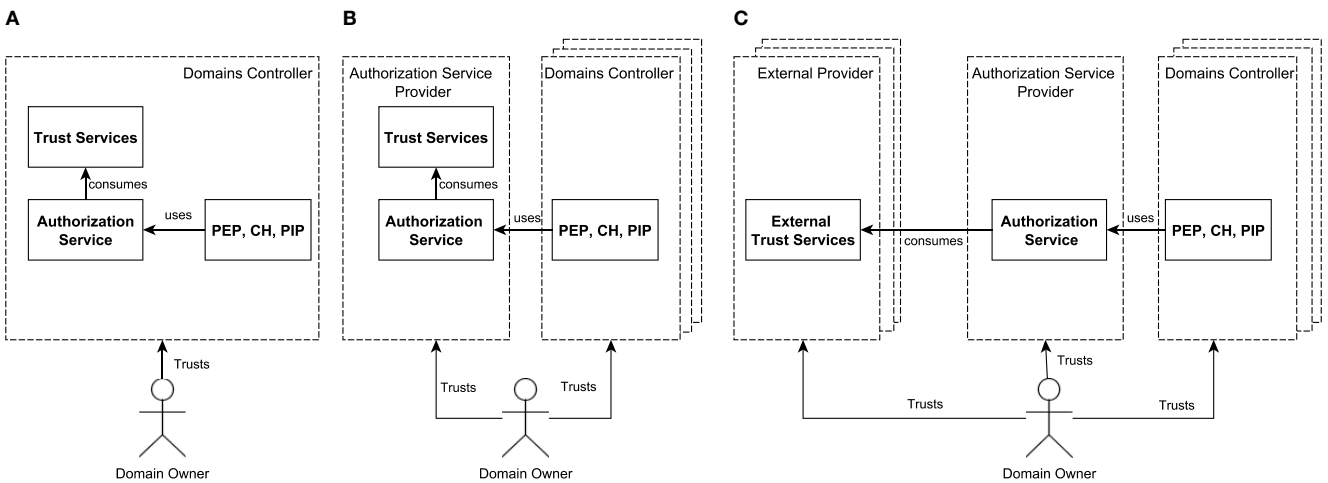

FIGURE 10 | Trust models and their impact on SAFAX deployment. **(A)** Centralized trust model. **(B)** Distributed trust model. **(C)** Distributed and delegated trust model.

that enforce the authorization decision (**Figure 10C**). However, in the distributed and delegated trust model, trust services are not only external to the authorization service but they can also be outside the control of the authorization service provider. As shown in **Figure 10C**, the **DO** trusts the authorization service to securely store its access control policies, configuration settings regarding the PDP and for proper evaluation of its access control policies. The **DO** trusts the **DC** to enforce the authorization decision determined by the authorization service. Moreover, the **DO** trusts the external trust services to provide accurate and complete information to the authorization service during policy evaluation. It is worth noting that the authorization service delegates part of policy evaluation to the external services even when there is no explicit trust relation existing between the authorization service and the external services. Indeed, this trust model empowers **DO**s with full control over policy evaluation in the sense that a **DO** can select the services that are used to evaluate its policies.

SAFAX supports all these three trust models, since at an architectural level we do not prescribe or constrain the deployment of the various SAFAX services. SAFAX facilitates such a variety of deployment configurations given that the individual services within the SAFAX framework are designed as loosely coupled services. The deployment of our prototype is based on the *Distributed and Delegated Trust Model* since it is the most open and extensible among the trust models. It allows our prototype to consume information from external trust sources as well as extend the functionality of the PDP. However, given that the various components are SAFAX are essentially web services, we can easily support the other two trust models by deploying the various services in different organizational perimeters. The deployment of a service does not have any impact on the SAFAX framework, because all the services are registered to the *Service Repository* service, which in turn provides different services the information needed to consume other services. In this manner, SAFAX framework can serve different organizations with specific trust models without any changes to the framework.

5.2. Lessons Learned from Deploying, Using, and Securing SAFAX

Since SAFAX can be used by multiple **DO**s and **DC**s, a basic authentication mechanism is provided by our prototype implementation in order to identity various entities. SAFAX processes and stores data from various entities: **DO**s, **DC**s, External Service Providers, and SAFAX Administrators. This information ranges from access control policies of **DO**s, configuration settings of a **DO**s PDP, access request logs from **DC**s, to global settings of the SAFAX framework. In addition, our implementation supports a *Guest* mode, through which unregistered users can configure a *Test PDP*, deploy access control policies and test access requests against their policies. Finally, SAFAX supports multiple users to act as a **DO** and configure access control policies in a collaborative manner and deploy them on a PDP that is shared between those users.

In order to regulate the actions that users can perform within SAFAX, we use SAFAX as the access control mechanism. In other words, SAFAX inherently plays the role of **DC** to control data stored within its **Domain**. We have developed the PEP, CH, and PIP specific for SAFAX (referred as SFX-*). Every action that a user wants to perform within SAFAX triggers an access request that is intercepted by the SFX-PEP. The request is enriched with attributes retrieved from SFX-PIP and then sent to the SAFAX-CORE service for evaluation. We specified a set of XACML policies that define the actions (e.g., create, view, edit, delete) users can perform on various SAFAX resources, essentially also playing the role of a **DO**. For instance, these policies are used to restrict the number of projects and demos users can create based on the groups they belong to and to limit the actions that guest users can perform.

Through this experience of securing SAFAX using SAFAX, we have gained an understanding of using SAFAX from the perspectives of both **DC**s and **DO**s. We have observed that the SFX-PIP plays a key role in the correct evaluation of policies since the native requests coming to the SFX-PEP from the various components of SAFAX do not contain all the information necessary to evaluate

the policies. The PIP needs to be aware of the various resources in its domain, the contextual information regarding those resources such as resource owners, resource types, project membership among others. In our prototype implementation, we support the two approaches described in Section 4 for attribute retrieval. In particular, attributes that are often used in policies, like resource owner and resource type, are fetched from the SFX-PIP when the XACML request is constructed. On the other hand, attributes that are checked in few specific conditions in the defined policies (e.g., number of projects and demos already created by a user) are fetched during policy evaluation.

We have observed that the **DO**s must be aware of the attributes that are supported by the **DC**s, in order to refer to them in their access control policies. Since in our case, the **DO** and the **DC** are a single entity, it is straightforward to align the various attributes. However, when a **DO** is subscribed to multiple **DC**s, it cannot be expected that they are aware of the syntax and semantics of various attributes in different **DC**s. For example, a **DO** may want to specify that a policy based on the resource's attribute such as *File Extension* for resources stored in two different clouds; however, the two cloud providers may use different attribute identifiers for the same attribute, which makes it infeasible for the PDP to match these different attributes (with different identifiers) to the attribute specified by the **DO**. This dependency can be avoided if the various cloud providers use a common syntax for the specification of various attributes. Another approach, which is currently used in `SAFAX` to remove this dependency between **DO**s and **DC**s, is by using a *policy alignment* service that is provided as part of the `SAFAX` framework. This service enables a semantic alignment between the concepts that can be used by different **DC**s based on the concept of similarity. More details regarding the policy alignment service can be found in Trivellato et al. (2009).

Overall, we have seen the advantages that `SAFAX` offers to both **DO**s and **DC**s in terms of ease of specification, maintenance, and evaluation of policies. Finally, the extensibility of `SAFAX` opens up the scope of access control to incorporate external trust sources in policy specification and evaluation.

6. Related Work

Two main streams of research can be identified in the literature, which aim to enable secure data sharing in cloud environments. The first stream stems from the fact that outsourcing data outside the trust domain reduces the control users have on their data. Solutions in this stream often rely on cryptographic schemes, which allow users to encrypt sensitive data and distribute keys to authorized users (Blundo et al., 2010; Wang et al., 2010; Yu et al., 2010; Ruj et al., 2011). A scheme often used by these solutions is hierarchical identity-based encryption (Boneh et al., 2005), which is a generalization of the identity-based encryption scheme (Shamir, 1985) that reflects the organizational hierarchy. However, these solutions have the intrinsic problems of identity-based policies and, thus, they are not able to address the challenges for cloud. Attribute-based encryption (ABE) (Sahai and Waters, 2005) has been proposed to enable encryption of sensitive information according to an attribute-based policy in such a way that only users with certain attributes (e.g., roles) can access the information. A number of variants of the ABE scheme have been proposed since its introduction. They range from extending its functionality (Wang et al., 2010; Li et al., 2011; Asim et al., 2012) to proposing schemes with stronger security proofs (Ibraimi et al., 2010). When applied to multi-cloud environments, the main limitation of most ABE-based schemes is that they require binding the attributes to a specific domain controller or a domain owner at encryption time. Therefore, data need to be re-encrypted every time a new domain owner joins the cloud. Moreover, ABE-based schemes implicitly assume the existence of a mechanism for determining the attribute keys that their users are entitled to receive. However, the attributes of a user may depend on other attributes or conditions determined by third parties. As a result, these schemes do not address the dynamics characterizing clouds and thus fail to meet challenge **C8**. Asim et al. (2012) present a dynamic ABE scheme that does not require the encryptor to bind the attributes to a specific domain controller or a domain owner at encryption time, enabling new users to decrypt the data without the need of re-encrypting it. Yet, cryptographic schemes require a key management mechanism that determines which user is entitled to receive the key to decrypt the data. As shown in Asim et al. (2012), authorization mechanisms (i.e., solutions in the second stream) are a viable solution for key management.

The second stream of solutions to regulate the sharing of sensitive information in cloud encompasses the use of authorization mechanisms. Several proposals (Hu et al., 2009; Alcaraz Calero et al., 2010; Almutairi et al., 2012; Wu et al., 2013) adopt Role Based Access Control (RBAC) to meet the dynamic, extensible, and highly configurable security requirements of clouds. Younis et al. (2014) propose AC3, an access control model for cloud environments that combines Task-Role Based Access Control (Oh and Park, 2003) with multilevel security. One major drawback of these approaches is the limited expressiveness of the policies that can be specified (challenge **C2**). In this work, we used XACML (OASIS, 2013) as policy language. This choice is twofold. First, XACML has become the *de facto* standard for access control (Liu et al., 2008), encouraging its adoption by a larger number of cloud providers. Moreover, it has been shown that attribute-based access control and, in particular, XACML (OASIS, 2013) provide a baseline for the development of authorization services for cloud environments (Takabi et al., 2010). The XACML reference architecture provides a blueprint, which needs to be followed by any implementation to be compliant with the standard. Several open-source XACML implementations including SUN-XACML[3], HERAS-AF (Dolski et al., 2007), XEngine (Liu et al., 2011), enterprise-java-xacml[4], and WSO2 Balana[5] are currently available. These implementations support different versions (and in some cases, multiple versions) of XACML. Some of these implementations, such as HERAS-AF and WSO2 Balana, are actively developed compared to other implementations. In addition, some implementations have been used as a backbone for XACML-based frameworks. For instance,

[3]http://sunxacml.sourceforge.net
[4]http://code.google.com/p/enterprise-java-xacml
[5]http://xacmlinfo.org/category/balana

Lazouski et al. (2014) extend WOS2 Balana for the enforcement of usage control policies in distributed systems, while TAS3 Trust PDP (Böhm et al., 2010) extends SUN-XACML for the evaluation of trust policies. The XACML reference architecture, however, is underspecified, and existing XACML implementations of XACML may have slight variations based on implementation choices. Nonetheless, a common characteristic of existing XACML implementations is that they are implemented as a monolithic component. Although the choice of implementing the XACML reference architecture as a monolithic component may be suitable for enterprise environment, it cannot fully address the key challenges for cloud environments. This problem is acknowledged and partially addressed by Laborde et al. (2009), who extend a core XACML PDP with loosely coupled domain-dependent modules that provide non-standard security information (such as new data types). However, the approach does not discuss the decoupling of the other components of the XACML reference architecture, and does not address the fact that UDFs are typically implemented within the PDP, thus limiting the extensibility of existing implementations. By contrast, SAFAX decouples the definition and implementation of UDFs from the PDP, thus allowing domain owners to determine from which source the information used for policy evaluation is fetched. Another approach for providing authorization as a service using XACML is proposed by Lorch et al. (2003). This solution addresses challenges **C2** and **C3** by providing a single point for deploying and updating policies, but does not address the other challenges to be faced in cloud, especially the ones related to policy evaluation.

7. Conclusion

In this paper, we have investigated the challenges of secure data sharing in multi-cloud environments. The XACML reference architecture provides the principles to address the identified challenges. However, existing XACML implementations are monolithic, thus resulting unsuitable for handling authorizations in multi-cloud environments. To address this limitation we have proposed SAFAX, an extensible XACML-based authorization framework tailored to the development of authorization services that are suitable for (multi) cloud environments. The main characteristic of SAFAX is that all its components are designed as loosely coupled services. This design choice makes SAFAX a viable solution as it provides the necessary flexibility, extensibility and scalability to meet the dynamic, extensible, and highly configurable security requirements of clouds. To demonstrate SAFAX, we have realized a prototype by implementing the SAFAX architecture as a web service. The trust model underlying an application domain has a significant impact on the deployment of an authorization service. We have reviewed a number of trust models typically being used in cloud environments and we showed that SAFAX can support them. Moreover, we discussed the lessons learned from deploying, using, and securing SAFAX.

Our current implementation of SAFAX uses the XACML PDP provided by HERAS-AF. This PDP supports the evaluation of policies specified in XACML v2.0. We plan to extend the evaluation capabilities with an additional PDP service to support XACML v3.0. Our implementation allows users to verify certain types of trust information in their policies, but other types of trust (e.g., KPI, risk) and context (e.g., location) information may be needed for making access decisions. In this direction, our implementation can be easily extended with additional trust services.

Acknowledgments

This work has been partially funded by the EDA project IN4STARS2.0, the ITEA2 projects FedSS (No. 11009) and M2MGrid (No. 13011), the EU FP7 project AU2EU, and the Dutch national program COMMIT under the THeCS project.

References

Alcaraz Calero, J., Edwards, N., Kirschnick, J., Wilcock, L., and Wray, M. (2010). Toward a multi-tenancy authorization system for cloud services. *IEEE Secur. Priv.* 8, 48–55. doi:10.1109/MSP.2010.194

Almutairi, A., Sarfraz, M., Basalamah, S., Aref, W., and Ghafoor, A. (2012). A distributed access control architecture for cloud computing. *IEEE Software* 29, 36–44. doi:10.3389/fpsyg.2013.00200

Asim, M., Ignatenko, T., Petkovic, M., Trivellato, D., and Zannone, N. (2012). "Enforcing access control in virtual organizations using hierarchical attribute-based encryption," in *Proceedings of International Conference on Availability, Reliability and Security* (Prague: IEEE), 212–217.

Blundo, C., Cimato, S., De Capitani Di Vimercati, S., De Santis, A., Foresti, S., Paraboschi, S., et al. (2010). Managing key hierarchies for access control enforcement: heuristic approaches. *J. Comput. Secur.* 29, 533–547. doi:10.1016/j.cose.2009.12.006

Böhm, K., Etalle, S., den Hartog, J., Hütter, C., Trabelsi, S., Trivellato, D., et al. (2010). A flexible architecture for privacy-aware trust management. *J. Theor. Appl. Electron. Commer. Res.* 5, 77–96. doi:10.4067/S0718-18762010000200006

Boneh, D., Boyen, X., and Goh, E.-J. (2005). "Hierarchical identity based encryption with constant size ciphertext," in *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, LNCS 3494* (Aarhus: Springer), 440–456.

Buyya, R., Ranjan, R., and Calheiros, R. N. (2010). "Intercloud: utility-oriented federation of cloud computing environments for scaling of application services," in *Algorithms and Architectures for Parallel Processing* (Busan: Springer), 13–31.

Chadwick, D., Lievens, S., den Hartog, J., Pashalidis, A., and Alhadeff, J. (2011). "My private cloud overview: a trust, privacy and security infrastructure for the cloud," in *Proceedings of International Conference on Cloud Computing* (Washington, DC: IEEE), 752–753.

Dolski, S., Huonder, F., and Oberholzer, S. (2007). *HERAS-AF: XACML 2.0 Implementation*. Technical Report, University of Applied Sciences Rapperswil.

Gartner Inc. (2014). *Gartner Says Worldwide Traditional PC, Tablet, Ultramobile and Mobile Phone Shipments to Grow 4.2 Percent in 2014*. Available at: http://www.gartner.com/newsroom/id/2791017

Grobauer, B., Walloschek, T., and Stocker, E. (2011). Understanding cloud computing vulnerabilities. *IEEE Secur. Priv.* 9, 50–57. doi:10.1109/MSP.2010.115

Hu, L., Ying, S., Jia, X., and Zhao, K. (2009). "Towards an approach of semantic access control for cloud computing," in *Cloud Computing, LNCS 5931* (Beijing: Springer), 145–156.

Hühnlein, D., Roßnagel, H., and Zibuschka, J. (2010). "Diffusion of federated identity management," in *Sicherheit*, P-170, 25–36.

Ibraimi, L., Asim, M., and Petkovic, M. (2010). "An encryption scheme for a secure policy updating," in *Proceedings of International Conference on Security and Cryptography* (Athens: SciTePress), 399–408.

Kamvar, S. D., Schlosser, M. T., and Garcia-Molina, H. (2003). "The eigentrust algorithm for reputation management in P2P networks," in *Proceedings of the 12th International Conference on World Wide Web* (Budapest: ACM), 640–651.

Laborde, R., Cheaito, M., Barrere, F., and Benzekri, A. (2009). "An extensible XACML authorization web service: application to dynamic web sites access control," in *Proceedings of International Conference on Signal-Image Technology and Internet-Based Systems* (Marrakesh: IEEE), 499–505.

Lazouski, A., Mancini, G., Martinelli, F., and Mori, P. (2014). "Architecture, workflows, and prototype for stateful data usage control in cloud," in *Proceedings of International Workshop on Data Usage Management* (San Jose, CA: IEEE), 23–30.

Li, J., Wang, Q., Wang, C., and Ren, K. (2011). Enhancing attribute-based encryption with attribute hierarchy. *Mobile Net. Appl. Arch.* 16, 553–561. doi:10.1007/s11036-010-0233-y

Li, N., Winsborough, W. H., and Mitchell, J. C. (2003). Distributed credential chain discovery in trust management. *J. Comput. Secur.* 11, 35–86.

Liu, A. X., Chen, F., Hwang, J., and Xie, T. (2008). "XEngine: a fast and scalable XACML policy evaluation engine," in *Proceedings of International Conference on Measurement and Modeling of Computer Systems* (Annapolis, MD: ACM), 265–276.

Liu, A. X., Chen, F., Hwang, J., and Xie, T. (2011). Designing fast and scalable XACML policy evaluation engines. *IEEE Trans. Comput.* 60, 1802–1817. doi:10.1109/TC.2010.274

Lorch, M., Kafura, D., and Shah, S. (2003). "An XACML-based policy management and authorization service for globus resources," in *Proceedings of International Workshop on Grid Computing* (Phoenix: IEEE), 208.

Matheus, A., and Herrmann, J. (2012). Geospatial eXtensible Access Control Markup Language (GeoXACML) Version 1 Corrigendum. OGC Implementation Standard OGC 11-017, Open Geospatial Consortium.

OASIS. (2013). eXtensible Access Control Markup Language (XACML) Version 3.0. OASIS Standard.

Oh, S., and Park, S. (2003). Task-role-based access control model. *Inf. Syst.* 28, 533–562. doi:10.1016/S0306-4379(02)00029-7

Ruj, S., Nayak, A., and Stojmenovic, I. (2011). "Dacc: distributed access control in clouds," in *Proceedings of International Conference on Trust, Security and Privacy in Computing and Communications* (Changsha: IEEE), 91–98.

Sahai, A., and Waters, B. (2005). "Fuzzy identity-based encryption," in *Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, LNCS 3494* (Aarhus: Springer), 457–473.

Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *Computer* 29, 38–47. doi:10.1109/2.485845

Shamir, A. (1985). "Identity-based cryptosystems and signature schemes," in *Proceedings of International Cryptology Conference* (Santa Barbara, CA: Springer), 47–53.

Simone, A., Skoric, B., and Zannone, N. (2012). Flow-based reputation: more than just ranking. *Int. J. Inf. Technol. Decis. Mak.* 11, 551–578. doi:10.1142/S0219622012500113

Skoric, B., de Hoogh, S., and Zannone, N. (2014). Flow-based reputation with uncertainty: evidence-based subjective logic. *CoRR*abs/1402.3319.

Takabi, H., Joshi, J., and Ahn, G.-J. (2010). Security and privacy challenges in cloud computing environments. *IEEE Secur. Priv.* 8, 24–31. doi:10.1109/MSP.2010.186

Trivellato, D., Spiessens, F., Zannone, N., and Etalle, S. (2009). "Reputation-based ontology alignment for autonomy and interoperability in distributed access control," in *Proceedings of International Conference on Computational Science and Engineering* (Vancouver, BC: IEEE), 252–258.

Trivellato, D., Zannone, N., and Etalle, S. (2014). GEM: a distributed goal evaluation algorithm for trust management. *Theory Pract. Logic Program.* 14, 293–337. doi:10.1017/S1471068412000397

Trivellato, D., Zannone, N., Glaundrup, M., Skowronek, J., and Etalle, S. (2013). A semantic security framework for systems of systems. *Int. J. Coop. Inf. Syst.* 22, doi:10.1142/S0218843013500044

Wang, G., Liu, Q., and Wu, J. (2010). "Hierarchical attribute-based encryption for fine-grained access control in cloud storage services," in *Proceedings of Conference on Computer and Communications Security* (Chicago: ACM), 735–737.

Wu, R., Zhang, X., Ahn, G.-J., Sharifi, H., and Xie, H. (2013). "ACaaS: access control as a service for IaaS cloud," in *Proceedings of International Conference on Social Computing* (Alexandria, VA: IEEE), 423–428.

Younis, Y. A., Kifayat, K., and Merabti, M. (2014). An access control model for cloud computing. *J. Inf. Secur. Appl.* 19, 45–60. doi:10.1016/j.jisa.2014.04.003

Yu, S., Wang, C., Ren, K., and Lou, W. (2010). "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proceedings of Conference on Computer Communications* (San Diego, CA: IEEE), 1–9.

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.