Ontology-based Delegation of Access Control: An Enhancement to the XACML Delegation Profile

Malik Imran Daud, David Sánchez and Alexandre Viejo

UNESCO Chair in Data Privacy, Department of Computer Science and Mathematics,
Universitat Rovira i Virgili, Avda. Països Catalans, 26, 43007 Tarragona, Spain
{malikimran.daud, david.sanchez, alexandre.viejo}@urv.cat

Abstract. Delegation of access control (i.e. transferring access rights on a resource to another tenant) is crucial to efficiently decentralize the access control management in large and dynamic scenarios. Most of the delegation methods available in the literature are based on the RBAC or ABAC models. However, their applicability can be hampered by: i) the effort required to manage and enforce multiple roles for each delegatee (i.e. access roles and delegated roles) and ii) the efforts required to specify constraints for the enforcement of the delegated roles or policies. Moreover, the performance of these methods decreases proportionally as the number of users increase. To tackle these issues, we propose an ontology-based delegation framework that enhances the standard XACML delegation profile by modeling the delegation logics in an ontological way. By means of the ontology, the operations of delegation, verification and revocation of access rights can be performed on the workflow generated by instantiating the ontology classes and their interrelations according to the entities involved in the delegation. By exploiting these workflows, we propose a cost-effective algorithm that performs delegation operations without involving any human intervention.

Keywords: Security, Access Control, Delegation, ABAC, XACML, Ontology

1 Introduction

In the field of information security, access control management is a method to manage the access to the resources based on the identity of the users [1]. Delegation is one of the mechanisms to manage access control in a flexible way [2], wherein, users can transfer their access rights to other entities on a particular resource. Most of the delegation mechanisms are based on the role-based access control (RBAC) model [3], where access rights are delegated in the form of roles. Few models rely on the attribute-based access control (ABAC) model [4] for delegation, where delegation is managed by using policies instead of roles.

XACML delegation profile [5] is one of the delegation mechanisms that is based on the ABAC model. In this mechanism, access rights on the resources are delegated in the form of policies. In the XACML profile, reduction is a process that is performed to validate the authenticity of the issuer of the policy. In this process, a graph of policies is generated as a result of each access request for a resource, which con-

tains the hierarchy of the delegated policies. To generate a policy graph, the attributes of the *access request* are searched (i.e. the delegatee and requested resource) within the policy delegated to the requester and, then, edges between that policy and its delegated policy nodes are created by matching the attributes of the entities within the hierarchy of the delegated policies (attributes are the delegatee and its delegator). Then, the path of the graph, which connects the owner of a resource and the requester with all the intermediate delegators, is checked in order to verify the authenticity of the delegated policy of the requester. As a result, decisions are made in the form of permit or deny access to the resource.

In this approach, the method to evaluate an access request, that is, generating a policy graph and finding attributes within all policies for each *access request*, is a costly solution in terms of performance. Figure 1 illustrates a graph generated as a result of access request. The connection between the delegated policies represents the flow of the delegation, whereas the edges determine the delegation decision, that is, the policy permits (PP) the delegation or denies (DP) it to the other policy.

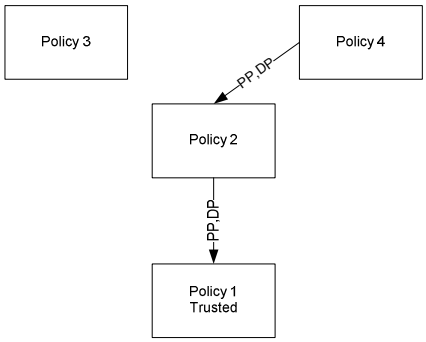


Fig. 1. XACML delegation graph [5]

In the XACML delegation profile, the delegated policies defined by the delegators on a common resource are maintained in a policy set. Thus, to process an access request for a resource, this policy set is searched to find the required policies in order to generate a flow of delegation in the graph of policies. This can result in a serious performance overhead in a large scale environment dealing with a large number of delegatees during the following actions: i) searching within a large number of delegated policies on an access request, and ii) generation of the graph on each access request. Moreover, the delegation graph is connected to the trusted policy, which differentiates with other policies with the omitted policy issuer element, and there is no mechanism to validate this trusted policy as the fields can easily be forged.

1.1 Contributions

To tackle these issues, in this paper we propose an ontology-based delegation framework that enhances the standard XACML delegation profile by modeling the delegation logics in an ontological way. In contrast to previous methods [6-8], the

specification of constraints or authorization rules for each delegator and delegatee to enforce delegation is not required; instead the proposed system automatically manages delegation by using suitable algorithms (i.e. delegation and verification). This is attained by a formal representation of the delegation workflow, which is made by instantiating an ontology modeling the entities and their interrelations that are involved in the delegation process. Moreover, it does not require generating rules or policy graphs for each access request for a resource; instead the request is validated through the interrelations of the entities. Furthermore, in order to build trust, each workflow of the delegation is originated by the trusted policy that is digitally signed by the owner of the resource. We next summarize the main contributions of our work:

- We propose an ontology that models the entities involved in the delegation process, which includes subjects (i.e., delegators or delegatees), objects (resources or services), policies (document that translate delegated privileges) and their interrelations. This ontology facilitates to keep a track of who is delegating, what privileges on a resource are being delegated and also provides an intuitive solution to verify the attributes of the actors involved in the delegation.
- In contrast to the methods that verify the delegator's authority through roles [9], our system automatically verifies the authority through the attributes of the entities and the policy of the delegator by following the interrelations of the entities (represented as instances of the ontology) that lead to the trusted policy.
- Contrary to [8, 10], our system does not require the specification of rules, but it automatically enforces delegation and verifies the delegated authority by using a simple algorithm. In addition, our proposal does not require any additional rules to implement the delegated policies; instead, it automatically implements a delegated policy that combines the normal access policy and the delegated policy and also resolves possible policy conflicts.

The rest of the paper is organized as follows. Section 2 presents our ontology-based delegation framework. Section 3 provides a discussion on the scalability and performance of the proposed system. Section 4 includes related works. Finally, section 5 provides the conclusions and presents some lines of future work.

2 Ontology-based Delegation Framework

We have extended the XACML delegation profile by modeling and incorporating an ontology to model the access control and delegation workflow; this provides two main benefits: i) it is easy and intuitive to implement, since it clearly defines the semantics (i.e. knowledge) of the delegation process, and ii) the relations between the entities can be defined and interpreted in an automatic way [11]. This ontology models the knowledge related to the delegation process according to the entities involved in it (i.e. delegators, delegatees and resources) and their interrelations. According to this, the actual entities involved in a particular delegation process are instances of the generic classes modeled in the ontology and their interrelations represent the workflow of the delegation. Therefore, to evaluate an *access request* for a resource, the authority of the delegator, who has issued a policy to the requester, is verified by examining

the instance interrelations (which include the delegators' hierarchy, the delegatee and the delegated resource) instead of finding entities within the policies. With this mechanism, the request attributes are only matched with the instances of the entities and their policy is checked just once to get their related rules. By doing so, it avoids the overhead of repeatedly examining each policy set (in order to find entities involved in the delegation) and then generating a policy graph for each access request (as done in the XACML profile). Moreover, it is also an intuitive way to validate the attributes of a policy with the attributes of the entity instances (attributes can be the identities of the delegator and the delegatee and their privileges).

In our model, a user can be assigned two types of policies (as shown in figure 2): i) an *access policy* (a policy defined for a particular user in order to allow or deny the access to a given resource) and ii) an *administrative policy* (a policy that enables a user to issue access policies or delegated policies to other users). For example, in the access policy defined in figure 2(a), the administrator allows user *Alice* to use the services of *printer-1*, whereas in the administrative policy in figure 2(b), the administrator also delegates the possibility to issue policies on the same resource (*access or administrative policies*) to the users of the *employee* category. In the *rule* attribute, the delegator can define limitations on the delegation in order to grant a limited access on a given resource. Moreover, the owner of the resource can limit the delegation (i.e. the number of times or levels a resource can be delegated) by setting a value of an attribute of the policy, which is reduced at each delegation level and added to the *policy set* of the delegatee. The access rights cannot be further delegated if the attribute value reaches zero.

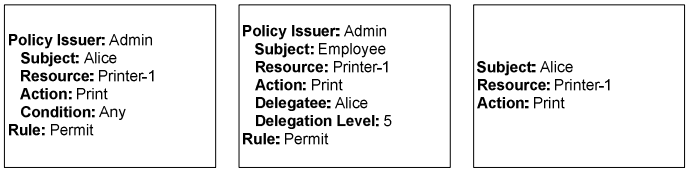


Fig. 2. Examples of policies and access request

The policies defined by the delegator for the different users on the same resource are managed in a document called a *policy set*. A *policy set* is a document written by an issuer, which contains several policies (*access* or *administrative policies*) and may have different rules for users on a common resource. In the XACML profile, a *policy set* is maintained with respect to the resource that contains the policies of all the delegators sharing privileges on a common resource. In contrast to XACML, in our model, each beneficiary of the delegation maintains a separate *policy set* for its resource in a distributed way. As a result of an *access request*, only those policies are checked according to the interrelations that are managed by the required delegators. For example, if a delegator *Alice* is managing two resources, then she maintains two *policy sets* of the delegated policies; as a result of an *access request* by the delegatee, only Alice's *policy set* (managed for intended resource) will be examined. The benefits of this approach are: i) decentralizing the *policy sets*, ii) incorporating delegation for

distributed environment, and iii) improving the process of verification of the delegation authority.

2.1 Modeling ABAC as an Ontology

In ABAC, there are three main entities that are involved in managing access control: *subjects*, *objects* and *policies*. Subjects are the owners of the resources and can control the access to their resource objects (e.g. data, services, applications or network resources) by writing *access policies* for other users. In these policies, the access rights on the resource objects are managed by defining policy rules for other subjects. The subjects and the objects are identified by their attributes, which are also used for taking authorization decisions in order to manage access rights. During the authorization, the policy rules are evaluated against the attributes of the subjects and the objects, and access decisions are taken based on this evaluation. In order to automatically manage the workflow of the delegation process, the attributes of the policies (shown in figure 2(a) and 2(b)) can be used to create interrelations within the entities by modeling them as ontology classes and their properties.

In the same way, subjects can delegate their access rights to others by using attributes of the entities and by defining delegation policies. Therefore, we can define a delegation workflow between subjects, objects and policies by modeling them in the ontology and, based on the workflow, we can also make delegation and authorization decisions. Hence, in our ontology, the entity types are represented as classes. The relationships among the entities are represented by directed edges that show a dependency between classes that are either a subclass or a property of the classes. Figure 3 depicts the proposed ontology that models the knowledge of the delegation process (i.e., entities and interrelationships).

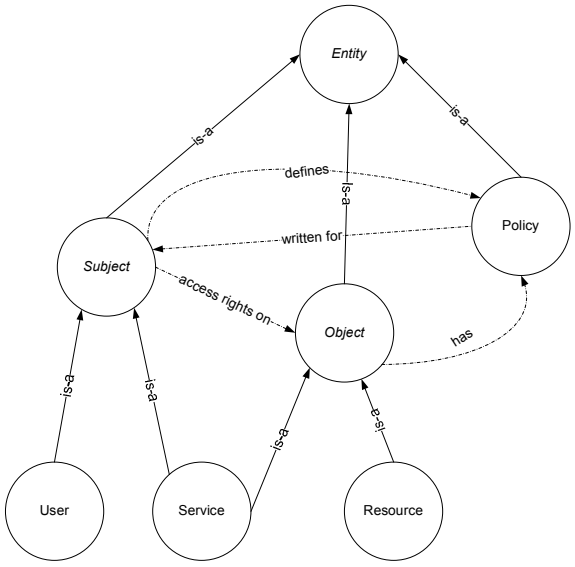


Fig. 3. Ontology representing the entities and interrelationships involved in delegation

The *entity* abstract class is the root of the ontology, which has three subclasses: *subject*, *object* and *policy*. A subject is an entity that may require access to the resource in order to accomplish her task; in general, that entity can be an organization, an employee, a department, or it can be any software-computing service. Therefore, subjects can be generally classified into two subclasses i.e. *user* and *service*. The *user* class models all types of subjects (mentioned above) except services, which are modeled in the *service* class. The attributes of the *user* class may vary depending on the type of the subject. For example, if a subject is an organization, then attributes can be the identifier of the organization, the organization name etc.; for a department, the attributes can be the department identifier, the department name or the organization it belongs to. Similarly, the services provided are modeled in *service* class. A *service* is a subclass of the *subject* class because it may also require resources in order to deliver its services and these services are identified and managed by their attributes (e.g. the service identity, the name of service or the provider of the service).

Subjects (e.g. users or services) can have or can define *access rights on* resources, which are modeled as *objects*. The relationship between the *subject* and the *object* classes are defined as ontological properties. *Objects* can be hardware resources (e.g. storage space, servers, etc.) or software resources (e.g. a web application, web service, etc.). Hardware resources are modeled as *resources* and they are managed by *subjects*, who are the owners. On the other hand, software resources are modeled as *services* that are also a subclass of the *object* class. Thus, *service* is a class that is inherited by both of the *subject* and the *object* classes, because, a service is treated as *subject* when it requires an access to the resources in order to accomplish its task and it is treated as an *object* when a user requires it as a service. The system can easily manage *service* instances according to their classification (i.e. as *subject* or *object*) from their attributes and their related policies, because a policy contains information about who is delegating to whom and what resources.

The *subject* and the *object* classes are linked to the *policy* class through their respective properties. In order to control the access, the subject *defines* policies for others to manage access rights on their resources (i.e. *object*). In practice, the attributes of subjects and objects are used in the *policy* class, and the access rights are delegated by defining related rules. The attributes of the *policy* class are same as mentioned in figure 2. The policy of the owner is different from the policies of other delegators, that is, this policy contains an attribute that is digitally signed by the owner and considered a *trusted policy*, which can be verified during the access request.

Ontology-based representation of the delegation workflow.

The ontology presented in figure 3 can be instantiated to represent the workflow of the delegated access rights. To do so, instances of the entity classes (i.e. delegators, delegatees and their policies) are created, and the delegation interrelations are managed based on their properties. In order to delegate access rights, each instance of the *subject* class, provided that it has delegation privileges, maintains a separate *policy set* that is stored in the local repository of the delegator and it is represented with the policy instance and linked to the instances of the delegatee *objects*.

An example of the delegation workflow generated through the proposed ontology is presented in figure 4. In this example, a *Dept1* (i.e. an instance of the *user* subclass) is the owner of two services (*Service-1* and *Service-2*) that are the instances of the *object* subclass *Service*. *Dept1* delegates its privileges on both services to the users *Alice*, *Bob* and *Alex* (who are also *subjects*) by defining policy sets *P1* and *P2* respectively. *Alice* and *Bob* share access rights on *Service-1* and have a common policy set *P1*, whereas *Dept1* maintains a separate policy *P2* for the user *Alex* with who shares access rights on *Service-2*. The policy set *P1* contains an *access policy* for *Alice* and an *administrative policy* for *Bob*, whereas, the policy set *P2* only contains an *access policy* for *Alex*. As a result of the *access policies*, *Alice* and *Alex* can only access *Service-1* and *Service-2* respectively (but cannot further delegate access rights); on the contrary, *Bob* is authorized to access and also to further delegate access rights to other users (due to *administrative policy*). In another level of delegation, *Bob* further delegates the access rights on *Service-1* to the users *Ted* and *Fred* by specifying a *policy set P4*.

For each delegation, the system creates instances of the entities involved in the delegation and links them automatically with each other as it is shown in figure 4. From this representation, the privileges of each accessing entity can be verified. In addition, the privileges of the delegation authority, who has issued her *access policy*, can also be validated from the delegation workflow.

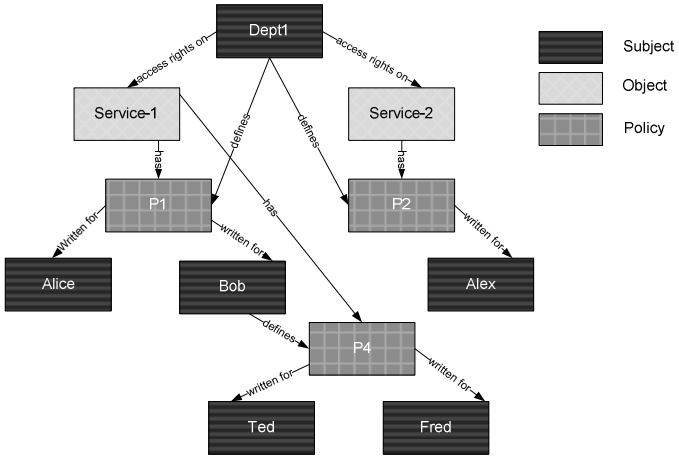


Fig. 4. Ontology-based representation of the delegation workflow

2.2 Workflow of the System

As shown in the previous section, during each delegation, instances of the involved entities are created together with the interrelations between the delegator and the delegated resource according to the knowledge modeled in the ontology; this forms a graph like workflow among the instances (as shown in figure 4). Then, as a result of a user's *access request*, the user's privileges are assessed and the authenticity of the

issuer of the policy (who has issued the user's *access policy*) is verified through the workflow generated from the ontology. To do so, the attributes of the *access request*, as shown in figure 2(c), are validated with the attributes of its immediate *access policies* through the policy instance linked with the requestor. If the attributes are matched, the system initiates an *administrative request* and the authenticity of the issuer of the policy is verified. To do so, the attributes of the issuer are verified from the delegator instances that are linked with the target *access policy*. Then, a chain of the *administrative policies* is generated at each level of the delegation to verify the authority at each level from the workflow graph (because each policy is issued by its predecessor delegator that also needs verification). As a result, a permit response is generated and access to the resource is granted in case the flow of the delegation is originated from the owner of the resource and a denial of access otherwise. Moreover, a policy is considered authentic if the hierarchy of policies leads to the trusted policy (a policy that is digitally signed and verified by the owner of the target resource or service) and there are no further policies connected to it in the workflow.

In the same way, a delegator, at any level of delegation, can revoke delegated privileges by generating a simple *revoke request*. Consequently, a chain of policies is verified (backward chaining beginning from the request initiator) and, if the request is genuine, all subsequent delegations are revoked by simply deleting instances from the workflow generated from the ontology.

The above process is formalized in Algorithm-1, which is invoked to verify the access rights of the requestor and to validate the delegator's authority that has granted the access rights. Line 1 determines the instance of the requestor from the workflow. Then, the access privileges of this requestor are determined (in line 5) from the policy stored locally in the database through the policy instance determined in line 4. Once the requestor has privileges (line 6), represented in the form of policy rule on the requested resource, then the authority of the delegator is verified. To do so, line 7 determines the instance of the delegator from the workflow. The same process is repeated (lines 3-14) for the chain of delegators until the owner of the resource is found in the workflow. Then, the signature of the owner is verified in lines 18-22 and access is granted to the resource provided the owner has digitally signed the policy and the delegator is verified.

Algorithm 1: *AccessRequest (subject_requestor, object_resource)*

```
1:    subject_instance ← subject_requestor
2:    delegator_verified=false
3:    while owner of object_resource is not found do
4:          get policy instance written for subject_instance
5:          read policy rule
6:          if rule is 'permit' then
7:              subject_instance ← get delegator instance that de-
               fined policy instance
8:              delegator_verified= true;
9:            else
10:                Deny Access
11:                delegator_verified=false
12:                if delegator_verified is 'false' then
13:                   end iteration in line 3
14:                end if
15:           end if
16:   end while
17:   if owner found and delegator is verified then
18:      signature ← verify owner's signature using her public key
```

```
19:       if signature is valid then
20:          Grant Access
21:       else   Deny Access
22:       end if
23:   else  Deny Access
24:   end if
```

2.3 Policy Conflicts

In the studied scenario, there can be the following situations in which two policies contradict the rules of each other:

1. A user has *access policies* issued by two different subjects on a common resource, where one policy permits the access to the resource while the other one denies it.
2. A user has two policies issued by two different subjects on a common resource, where one policy contains an *access policy* that only allows her to access to the resource, whereas the other policy contains an *administrative policy* that delegates access rights to issue policies on the same resource.

The policy conflicts mentioned above are automatically handled by the system without requiring any additional constraints or rule specifications. Conflict resolution is achieved by comparing the precedence of the issuers of the policies, that is, the delegation precedence of the conflicting delegators is determined from the delegation workflow. The preceding delegator is given a higher priority for the implementation of her policy because the upper level entities delegate privileges and can revoke them for the subsequent delegators. In case the delegators have same precedence level, the strictest rule of the policies of the delegators is implemented (e.g., denial dominates over permit). Notice that the determination of the precedence of the policy issuers does not incur any extra cost, since it can be determined from the delegation level attribute (discussed in section 2) of the policies of the delegators.

3 Performance Analysis

As mentioned in section 2, the proposed model improves the XACML delegation profile in following three ways: i) better management of the policies, ii) intuitive ontology-based generation delegation workflows iii) more efficient policy search while processing an access request. In order to quantify the efficiency improvement of the policy search, in this section we analyze and compare the computational complexity of the standard XACML profile with respect to our proposal.

In the XACML profile, on each *access request*, the system searches all the policies in a given policy set to get the policy of the requester and, in order to verify the authority, the credentials of the *access request* are matched with the elements of the policies. Therefore, in the worst case, the search requires to compare M policies (i.e. $O(M)$) and the verification of the authority requires matching M policies to generate the policy graph, which is also $O(M)$; thus, the overall cost of this process is $O(2M)$.

In contrast, our system can directly get the policy of the requester from the delegation workflow (which associates policies with the corresponding requester instance),

which is a constant cost ($O(1)$). Then, the authority of the delegator can be verified by following the workflow that, in the worst case, would take $\log_n N$ matches, where n is the number of delegator nodes who have delegated policies to the instance being verified, and N is the number of delegators in the delegation chain of the resource. As a result, the cost of our proposal is $O(\log_n N)$. Thus, as the number of delegators increases, this N becomes smaller then M, which proves that the proposed system results in performance gain over the XACML profile.

4 Related work

Several researchers have addressed the issues of access control management by proposing solutions that rely on delegation mechanisms. Most of these solutions [6-8, 12] have extended the role-based access control (RBAC) model [3] to incorporate delegation, wherein roles are delegated to other users in order to grant access privileges. In contrast, few models [13, 14] rely on attribute-based access control (ABAC) model [4] for delegation, where delegation is served by using policies instead of roles. ABAC is a better model than RBAC in terms of scalability (i.e. number of users), flexibility (i.e. easy to implement in a large scale environment) and better access control management (i.e. delegation by simply associating attributes to other users) [15, 16].

In the following, we discuss delegation frameworks that rely on the ABAC model, and specifically on the XACML profile, for delegation of access privileges. Xu et al. [13] proposed a delegation model based on the XACML standard and also extended this standard by introducing roles as the attributes of the users (by using roles as attributes of the policy extends ABAC with the capabilities of the RBAC model). In this method, roles are assigned to the delegators in order to enable them to delegate privileges that can further be delegated to other users. In this mechanism, the authority of delegators is verified from the role hierarchy that is generated as a result of each resource request, which can be an expensive solution in terms of performance (i.e. searching each role and plotting their graph on each request). In XACML based approach [14], a delegator can delegate limited privileges by defining a policy, and the system creates a new policy by updating an existing policy of a delegatee. In order to enforce delegation, a locally maintained system performs authorization of delegated policies. However, this system is customized for a specific application (i.e. account management) and cannot serve in large scale environment that has a large number of users.

Even though ontologies have been used to manage access control, ontology-based delegation of access rights have not been properly considered so far. For example, Choi et al. [17] proposed ontology-based context-aware access control model that manages context information of the users in an ontological way. The access decisions are made through the semantic analysis of the access rights of the service provider and the users. In this model, the access request of the user is analyzed and the access decision is made based on the context information of user's authentication and its access rights managed in form of security policy for a given resource. In this approach, separate policies are maintained for the administrator (i.e. service provider) and the user.

However, this may not be feasible for the delegation of access rights as a delegatee will have to maintain two policies, which are administrative policy being delegator of inherited access rights and the delegated policy being delegated. This may result in policy conflict, which will require special handling.

5 Conclusions and future work

In this paper, we proposed an access control delegation framework that enhances the XACML delegation profile by intuitively modeling the delegation process as an ontology-based workflow. Contrary to the XACML profile, the policies of the delegators are self-managed. As a result, only those policies that are related to the intended delegators and delegatees are analyzed, thus reducing the cost of searching entities within the policies. Moreover, it does not require generating a graph on each *access request* because the delegation workflow is automatically generated (by instantiating the ontology classes for the subject, object and policy) on each transfer of privileges by the delegator.

As future work, we plan to implement this framework in real and complex systems in which the delegation of access control has a prominent importance. For example, Cloud computing, in which the access to the outsourced resources should be managed for heterogeneous entities located at different places, and Social Networks, where users can delegate the access to their resources (e.g. messages, images, etc.) according to the attributes of their contacts and to revoke them accordingly. For this purpose, our general ontology can be extended to incorporate the entities and attributes involved in those platforms and delegation management algorithms can be tailored to implement their casuistry.

Acknowledgements and disclaimer

This work was partly supported by the European Commission under FP7 project Inter-Trust and H2020 project CLARUS, by the Spanish Ministry of Science and Innovation (through projects CO-PRIVACY TIN2011-27076-C03-01 and ICWT TIN2012-32757) and by the Government of Catalonia (under grant 2014 SGR 537). This work was also made possible through the support of a grant from Templeton World Charity Foundation. The opinions expressed in this paper are those of the authors and do not necessarily reflect the views of UNESCO of the Templeton World Charity Foundation.

References

1. Ferraiolo, D.F., Kuhn, R.D., Chandramouli, R.: Role-Based Access Control, Second Edition. Artech House, Inc. (2007)
2. Wang, Q., Li, N., Chen, H.: On the Security of Delegation in Access Control Systems. In: Jajodia, S., Lopez, J. (eds.) Computer Security - ESORICS 2008, vol. 5283, pp. 317-332. Springer Berlin Heidelberg (2008)

3. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-Based Access Control Models. Computer 29, 38-47 (1996)
4. Hu, V.C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to Attribute Based Access Control (ABAC) Definition and Considerations. NIST Special Publication (2014)
5. XACML v3.0 Administration and Delegation Profile Version 1.0. vol. 3.0. OASIS (2009)
6. Ruan, C., Varadharajan, V.: Dynamic delegation framework for role based access control in distributed data management systems. Distrib Parallel Databases 32, 245-269 (2014)
7. Sohr, K., Kuhlmann, M., Gogolla, M., Hu, H., Ahn, G.-J.: Comprehensive two-level analysis of role-based delegation and revocation policies with UML and OCL. Inf. Softw. Technol. 54, 1396-1417 (2012)
8. Wainer, J., Kumar, A., Barthelmess, P.: DW-RBAC: A formal security model of delegation and revocation in workflow systems. Information Systems 32, 365-384 (2007)
9. Ahn, G.-J., Mohan, B., Hong, S.-P.: Towards secure information sharing using role-based delegation. Journal of Network and Computer Applications 30, 42-59 (2007)
10. Wainer, J., Kumar, A.: A fine-grained, controllable, user-to-user delegation method in RBAC. Proceedings of the tenth ACM symposium on Access control models and technologies, pp. 59-66. ACM, Stockholm, Sweden (2005)
11. Carminati, B., Ferrari, E., Heatherly, R., Kantarcioglu, M., Thuraisingham, B.: A semantic web based framework for social network access control. Proceedings of the 14th ACM symposium on Access control models and technologies, pp. 177-186. ACM, Stresa, Italy (2009)
12. Gusmeroli, S., Piccione, S., Rotondi, D.: A capability-based security approach to manage access control in the Internet of Things. Mathematical and Computer Modelling 58, 1189-1205 (2013)
13. Xu, M., Wijesekera, D.: A role-based XACML administration and delegation profile and its enforcement architecture. Proceedings of the 2009 ACM workshop on Secure web services, pp. 53-60. ACM, Chicago, Illinois, USA (2009)
14. Seitz, L., Rissanen, E., Sandholm, T., Firozabadi, B.S., Mulmo, O.: Policy administration control and delegation using XACML and Delegent. In: Grid Computing, 2005. The 6th IEEE/ACM International Workshop on, pp. 6 pp., (2005)
15. Coyne, E., Weil, T.R.: ABAC and RBAC: Scalable, Flexible, and Auditable Access Management. vol. 15, pp. 14-16. IT Professional (2013)
16. Priebe, T., Dobmeier, W., Kamprath, N.: Supporting attribute-based access control with ontologies. In: Availability, Reliability and Security, 2006. ARES 2006. The First International Conference on, pp. 8 pp., (2006)
17. Choi, C., Choi, J., Kim, P.: Ontology-based access control model for security policy reasoning in cloud computing. J Supercomput 67, 711-722 (2014)