An XACML-Based Privacy-Centered Access Control System

Claudio A. Ardagna
Università degli Studi di Milano
26013 Crema - Italy
claudio.ardagna@unimi.it

Sabrina De Capitani di Vimercati
Università degli Studi di Milano
26013 Crema - Italy
sabrina.decapitani@unimi.it

Stefano Paraboschi
Università di Bergamo
24044 Dalmine - Italy
parabosc@unibg.it

Eros Pedrini
Università degli Studi di Milano
26013 Crema - Italy
eros.pedrini@unimi.it

Pierangela Samarati
Università degli Studi di Milano
26013 Crema - Italy
pierangela.samarati@unimi.it

ABSTRACT

The widespread diffusion of the Internet as the platform for accessing distributed services makes available a huge amount of personal data, and a corresponding concern and demand from users, as well as legislation, for solutions providing users with form of control on their data. Responding to this requirement raises the emerging need of solutions supporting proper information security governance, allowing enterprises managing user information to enforce restrictions on information acquisition as well as its processing and secondary use. While the research community has acknowledged this emerging scenario, and research efforts are being devoted to it, current technologies provide still limited solutions to the problem.

In this paper, we illustrate our effort in pursuing the goal of bringing information security governance restrictions deployable in current organizational contexts. Considering the large success and application of XACML, we extend the XACML architecture and modules complementing them with functionalities for effective credential-based management and privacy support. Our proposal combines XACML with PRIME, a novel solution supporting privacy-aware access control, resulting in an infrastructure that provides the flexible access functionality of XACML enriched with the data governance and privacy features of PRIME.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*Security, integrity, and protection*; H.4 [**Information Systems Applications**]: Miscellaneous; K.4.1 [**Computer and Society**]: Public Policy Issues—*Privacy, regulation*

WISG'09, November 13, 2009, Chicago, Illinois, USA.

General Terms

Design, Security

Keywords

Access Control, Data Governance, Privacy, XACML

1. INTRODUCTION

The huge success of the Web as a platform for the distribution of services and dissemination of information makes the protection of users' privacy a fundamental requirement. Privacy issues affect different aspects of today Internet transactions, and controlling access to private information plays a crucial role in providing privacy guarantees [13]. Although considerable work has been performed in the field of access control for distributed services [2, 3, 4, 15], available access control mechanisms are at an early stage from a privacy protection point of view. This situation reflects the fact that attention on security requirements has been mostly focused on addressing server-side security concerns (e.g., communication confidentiality, unauthorized access to services, data integrity). Here, we focus on the development of a privacy-aware access control system that takes into consideration both the client and the server sides. At the client side, we provide to the users the ability to preserve their privacy and to the enterprises the technical means to support the governance of privacy-sensitive data. Our approach aims at contributing to the development of a complete solution for security governance focusing on the support of user privacy.

An environment aiming at providing users with a private and secure way for using e-services should support at least the following basic requirements.

- *Privacy.* A digital identity solution should be respectful of the users' rights to privacy and should not disclose and process personal information without explicit consent.
- *User-driven constraints.* In addition to traditional server-side access control rules, users should be able to specify data handling constraints and restrictions about possible secondary use of their information once released to external parties.
- *Credential support.* Access control should support the

definition and enforcement of credential-based authorization policies.

- *Minimal disclosure.* Service providers must require the least set of credentials needed for service provision, and users should be able to provide credentials selectively, depending on the online services they wish to access.
- *Interactive enforcement.* A new way of enforcing the access control process should be defined based on a negotiation protocol aimed at establishing the least set of information that a user has to disclose to access a specific service.
- *Anonymity support.* As a special but notable case of minimal disclosure, many services do not need to know the real identity of a user. Pseudonyms, multiple digital identities, and even anonymous accesses must be adopted when possible.
- *Legislation support.* Privacy-related legislation is becoming a powerful driver towards the adoption of digital identities. The exchange of identity data should not violate government legislations, such as the Data Protection Directive in the European Union, or the Health Insurance Portability and Accountability Act (HIPAA) or Gramm-Leach-Bliley Act (GLB) in the USA.

Our reference scenario is a distributed infrastructure that includes *users* who request access to online services provided by *servers*. Servers may collect personal information of the users for evaluating access control policies, and users may pose restrictions on how this personal information should be managed.

An access control system monitors access requests and implements regulations (*policies*), which establish who can, or cannot, execute which actions on which resources [11]. In the scenario we are considering, traditional access control solutions result limiting and do not satisfy all the requirements mentioned above. As a matter of fact, access control systems allow the specification of policies with reference to generic attributes/properties of the parties and the resources involved [1, 4] and even if many access control proposals support privacy and trust negotiation, current available solutions do not provide support for privacy policies. XACML (eXtensible Access Control Markup Language) [4], the result of an OASIS standardization effort, represents today the most effective and accepted solution for controlling access in distributed environments. XACML proposes an XML-based language to express and interchange access control policies, and defines both an architecture for the evaluation of policies and a communication protocol for message exchange. Although XACML is largely adopted and is considered a reference solution, it supports neither privacy features nor the effective evaluation and specification of credential restrictions in the policies.[1] Also, it imposes that the data of the users be available at access request time, thus lacking support for negotiation/dialog between users and servers.

[1]Version 3.0 of XACML, currently (July 2009) in the preliminary "first draft" status, provides a privacy profile, which is however limited, consisting only of a few requirements and the explicitation of two attributes. Credential support is limited to the evaluation of attributes issuer, time, and date associated with certificates.

The consideration of privacy and data governance issues introduces the need to rethinking authorization policies and models, and the development of new paradigms for access control and authorization specification and enforcement. Two major issues need to be addressed. First, access control needs to operate even when interacting parties wish to disclose limited or no information about themselves. Second, data collected/released during access control, as well as data stored by the different parties, may contain sensitive information on which privacy policies need to be applied (data handling) and should therefore be protected. These issues have been investigated within PRIME (Privacy and Identity Management for Europe) [10], a large-scale EU-funded research project just arrived at its completion, which aimed at developing a privacy-enhancing Identity Management System that protects the personal information of the users and provides a framework that could be smoothly integrated with current architectures and online services. The goal of the PRIME project was to empower users, providing them with solutions to retain control over their personal information in interactions with other parties, also imposing constraints on subsequent data handling and secondary use. In the PRIME vision, the user should have control of personal information and negotiate its disclosure for access to a service. The result of such a negotiation is an agreement between the user and the service provider, whereby the provider collects personal data for a stated, legitimate purpose and under agreed conditions of use. The PRIME project has developed a privacy-aware access control system, which supports privacy requirements and extends traditional access control functionalities with support for data handling [1]. Notwithstanding the significant benefits of PRIME and of its access control implementation, PRIME has shown limited appeal to those complex business scenarios with stable legacy systems and database-centered architectures, with an already existing, well integrated, access control solution. Nowadays, the same business scenarios have shown some use of XACML for controlling access to data/resources in distributed settings, but they are reluctant to change their infrastructures to integrate the PRIME solution and its privacy functionalities.

The work presented in this paper, within the PrimeLife (Privacy and Identity Management in Europe for Life) project [9] which is the successor of PRIME, aims at providing a solution to the above issues by integrating the XACML and the PRIME architecture. The motivation is to build a flexible framework that exploits the advantages of XACML in terms of access control and scalability, and the advantages of PRIME in terms of data governance and privacy. The integration of the XACML access control within the PRIME architecture permits to put at use the PRIME features in existing business scenarios. By reducing the amount of required changes to the business and technological infrastructure, our solution provides effective deployment of privacy-support features in current information systems. Our solution also shows the flexibility of the XACML standard and of its implementation, and can serve as a guideline for others interested in extending XACML for capturing different protection requirements.

The remainder of this paper is organized as follows. Section 2 illustrates the XACML language and architecture. Section 3 presents the PRIME access control system and Section 4 describes its integration with XACML. Section 5

illustrates some aspects on the implementation. Section 6 discusses related work. Finally, Section 7 presents our conclusions.

2. EXTENSIBLE ACCESS CONTROL MARKUP LANGUAGE (XACML)

XACML [4] is an XML-based language for expressing and interchanging access control policies. The language offers the functionalities of most security policy languages and has standard extension points for defining new functions, data types, policy combination logic, and so on. In addition to the language, XACML defines both an architecture for the evaluation of policies and a communication protocol for message interchange. The main functions offered by XACML can be summarized as follows.

- *Policy combination.* XACML provides a method for combining policies independently specified. Different entities can then define their policies on the same resource. When an access request on that resource is submitted, the system takes into consideration all the applicable policies.
- *Combining algorithms.* Since XACML supports the definition of positive and negative authorizations, there is the need for a method for reconciling independently specified policies when their evaluation is contradictory. XACML supports different combining algorithms, each representing a way of combining multiple decisions into a single decision.
- *Attribute-based restrictions.* XACML supports the definition of policies based on generic properties (attributes) associated with subjects (e.g., name, address, occupation) and resources (e.g., creation date, type). XACML includes some built-in operators for comparing attribute values and provides a method for adding nonstandard functions.
- *Policy distribution.* Policies can be defined by different parties and enforced at different enforcement points. Also, XACML allows one policy to contain, or refer to, another.
- *Implementation independence.* XACML provides an abstraction layer that isolates the policy-writer from the implementation details. This guarantees that different implementations operate in a consistent way, regardless of the specific implementation.
- *Obligations.* XACML provides a method for specifying actions, called *obligations*, which must be fulfilled in conjunction with the policy enforcement, after the access decision has been taken.

Figure 1 illustrates the XACML working and the data flow in the access control evaluation. Access control works as follows. The requester sends an access request to the *Policy Enforcement Point* (PEP) module (step 2) which in turn send it to the *Context Handler*. The Context Handler translates the original request into a canonical format, called *XACML request context* (step 3) and send it to the *Policy Decision Point* (PDP) (step 4). The PDP identifies the applicable policies among the ones stored at the *Policy Administration Point* (PAP) and retrieves the attributes required

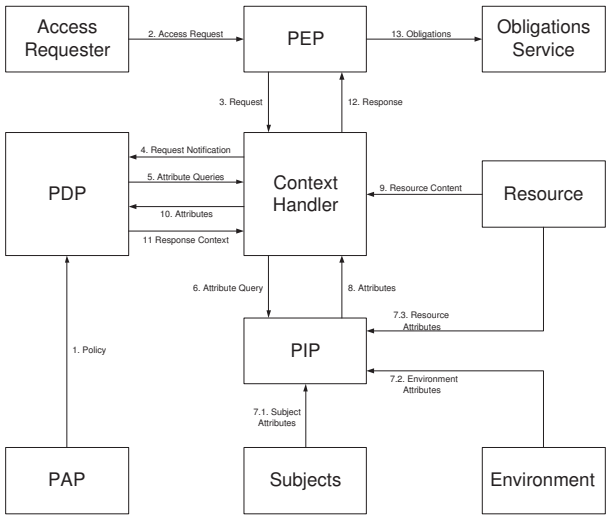

Figure 1: Overview of the XACML data flow [4]

for the evaluation through the Context Handler (steps 5-10). If some attributes are missing, the context handler queries the *Policy Information Point* (PIP) module for collecting them. The PIP provides attribute values about the *subject*, *resource*, and *environment*. To this purpose, the PIP interacts with the *subjects*, *resource*, and *environment* modules. The *environment* module provides a set of attributes that are relevant to take an authorization decision and are independent of a particular *subject*, *resource*, and *action*. The PDP evaluates the policies against the retrieved attributes, and returns the *XACML response context* to the Context Handler (step 11). The Context Handler translates the XACML response context to the native format of the PEP and returns it to the PEP together with an optional set of obligations (step 12). The PEP fulfills the obligations (step 13), and grants or denies the request according to the decision in the response context.

The XACML canonical form of the request/response managed by the PDP allows the definition and analysis of policies without taking into account the details of the application environment. Any implementation has to translate the attribute representations in the application environment (e.g., SAML, .NET, Corba [8]) into the XACML context. For instance, an application can provide a SAML [7] message that includes a set of attributes characterizing the subject of the access request. This message is translated in the XACML canonical form and, analogously, the XACML decision is translated in the SAML format.

3. PRIME ACCESS CONTROL SYSTEM

The PRIME privacy-aware access control system deals with five main key aspects: *i) resource representation,* to specify access control requirements on resources, in terms of available *metadata* describing them; *ii) subject identity,* to specify access control conditions on the subject requesting access and its personal information; *iii) secondary use,* to allow users to define restrictions on how their information will be used and processed by external parties after its release; *iv) context representation,* to provide contextual information in a standard format for the evaluation of policy conditions;

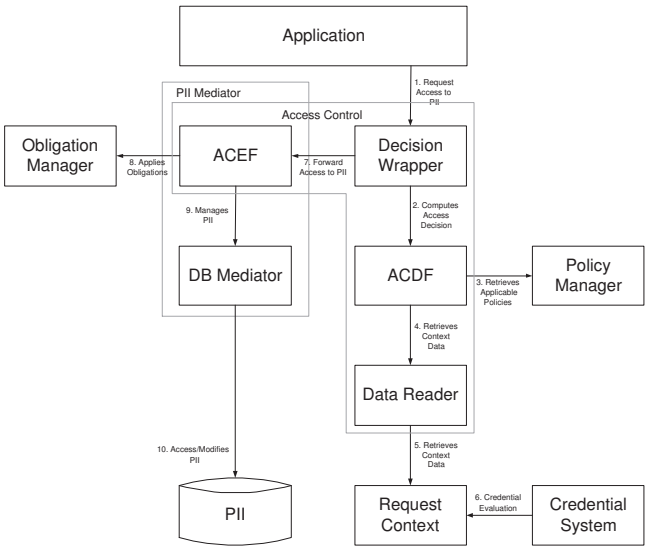

Figure 2: The PRIME Access Control Architecture

v) ontology integration, to exploit the Semantic Web and to allow the definition of access control rules based on generic assertions defined over concepts in the ontologies.

Figure 2 illustrates the architecture of PRIME and the flow of an access control evaluation (steps 1-6) and enforcement (steps 7-10). The architecture is composed of the following main components.

- *Decision Wrapper*: responsible for driving the access control policy evaluation and enforcement.
- *Access Control Decision Function (ACDF)*: responsible for taking access decisions for all access requests directed to resources (personal identifiable information (PII) or services). It is the core component of the PRIME access control implementation.
- *Policy Manager*: responsible for managing the overall policy life cycle by providing functions for administering policies. It also provides filtering functionality over the responses to be returned to the counterpart (user or service), to restrict the release of sensitive information related to the policy itself.
- *Request Context*: responsible for managing all contextual information; it stores all the data and credentials released by a user in a given session.
- *Data Reader*: responsible for abstracting the communication between the ACDF and the Request Context.
- *Credential System*: responsible for credential verification.
- *Access Control Enforcement Function (ACEF)*: responsible for enforcing access control decisions by mediating all accesses to resources and allowing them only if they are authorized by ACDF.
- *Obligation Manager (OM)*: responsible for managing, scheduling, enforcing, and monitoring privacy obligations. Obligations are actions that have to be performed either after an access has been granted or in the future, based on the occurrence of well defined events, e.g., time-based events or context-based events.
- *PII*: database storing all personal identifiable information.
- *DB Mediator*: responsible for abstracting the communication between the ACEF and the PII repository.

In the following, we concentrate our discussion on the ACDF component. The ACDF component is responsible for taking access decisions for all access requests directed to resources, by retrieving and evaluating all access control and data handling policies applicable to a request. Access control policies govern access to and release of resources managed by a party [11]. Data handling policies define how data (personal information) will be (or should be) dealt with at the receiving parties [1, 15]. An access request is modeled as a 4-uple of the form ⟨*subject, action, object, purpose*⟩, where *subject* is the optional identifier/pseudonym of the requester, *action* is the action that is being requested, *object* is the resource on which the requester wishes to perform the action, and *purpose* is the purpose (or a set of purposes) for which the access is requested. The ACDF produces the final response possibly combining the access decisions coming from the evaluation of different policies. The ACDF can then return three different decisions: *i) Yes*, the request can be granted; *ii) No*, the request must be denied; *iii) Undefined*, current information is not sufficient to determine whether the request can be granted or denied. In this case, additional information is needed and the counterpart will be asked to provide such an information.

As illustrated in Figure 2, the ACDF mainly interacts with the Request Context component. This component keeps track of all contextual information, combines information from various context sources, and describes new contextual information from this aggregation. Note that, the communication between ACDF and Request Context components is mediated by a façade component, called Data Reader. The Data Reader component abstracts the process of retrieving the information needed by the ACDF for the evaluation. This approach adds a level of isolation that guarantees a simple integration of the ACDF with different context formats or modules. The Request Context component also interacts with the Credential System, which is a credential verification component, in charge of verifying (possibly anonymous) credentials.

The evaluation flow of the ACDF is as follows. After receiving the access request, the ACDF: *i)* retrieves the access control policies by querying the Policy Manager (PM), *ii)* evaluates the access control policies, *iii)* collects the data handling policies attached to the object of the request, *iv)* evaluates the data handling policies, *v)* generate a single access decision.

The ACDF supports conditions to be evaluated on both certified data, issued and signed by authorities trusted for making the statement (credentials), and uncertified data, possibly signed by the data owner itself (declarations). Credentials and declarations relevant to an evaluation process are retrieved from the Request Context component. In case of credentials, the Request Context component retrieves the information needed by the ACDF by using the Credential System component.

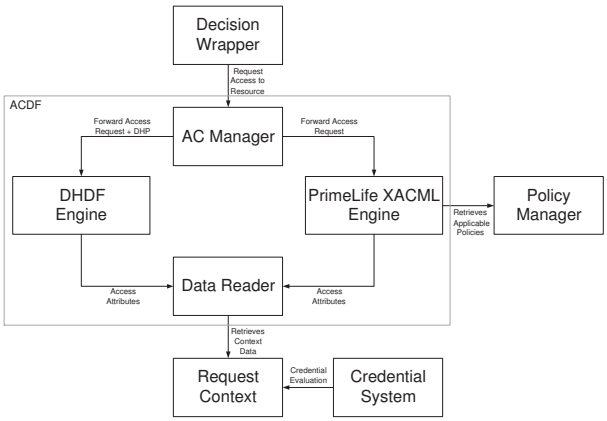

Figure 3: The PrimeLife access control architecture

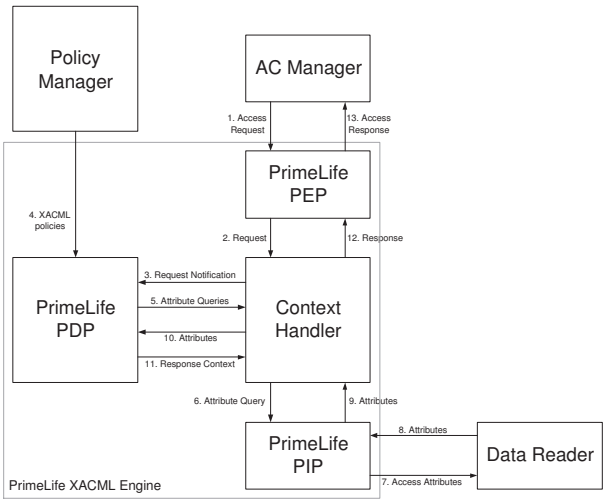

Figure 4: The PrimeLife XACML Engine data flow

4. THE PRIMELIFE ACCESS CONTROL SYSTEM

The PrimeLife project aims at providing a full identity management and data governance infrastructure allowing, on one side, users to maintain control over their data and to protect their privacy, and, on the other side, organizations to effectively provide their services respecting privacy regulations when processing private data of the users.

PRIME and XACML represent starting points for PrimeLife, and their careful integration provides important functionalities that neither XACML nor PRIME alone can provide. XACML, in fact, represents the most accepted and flexible access control language, but it does not provide effective support for privacy. PRIME, instead, provides privacy functionalities, but its impact on real world is limited by the fact that PRIME adoption in existing businesses would require important changes to legacy systems. Taking into account this analysis, PrimeLife decided to integrate the XACML evaluation and enforcement with the PRIME privacy functionalities. By integrating XACML into the PRIME architecture, we have all the advantages of the XACML-based solution (e.g., extendibility, flexibility), enriched with support for credentials, dialog between parties, incremental release of data (e.g., interactive enforcement), and data handling.

To enable coexistence of PRIME and XACML, the PrimeLife architecture employs two independent modules that have separate duties. For the sake of clarity, we use PrimeLife XACML Engine to denote the enhanced XACML Engine, and Data Handling Decision Function (DHDF) Engine to denote the modified version of the PRIME ACDF (see Figure 2). Specifically, the PrimeLife XACML Engine is devoted to the evaluation and enforcement of access control, while DHDF provides privacy and data handling functionalities. Differently from ACDF in the PRIME solution, DHDF does not implement a complete privacy-aware access control system, but rather it is responsible for the management and evaluation of data handling policies only.

The integration exploits a mechanism that the PrimeLife XACML architecture can use to freely access any external repository via a refined PIP component (see Section 2). The context information stored in the Request Context, and produced during the interaction with the requester, is an example of such repositories. PrimeLife XACML can then access the Request Context, to evaluate its policies, by means of a standard PIP extension, and can use the PRIME architecture to negotiate access and evaluate credentials.

Figure 3 depicts the resulting architecture. When an access control decision needs to be taken, the Decision Wrapper component forwards the access request to the AC Manager, together with the possible relevant data handling policies attached to the requested resources. The AC Manager, in charge of taking the final decision by combining the PrimeLife XACML access control and the DHDF data handling evaluation results, forwards the request to both the PrimeLife XACML Engine and the DHDF Engine. DHDF receives the request with the associated data handling policies, while PrimeLife XACML accesses the Policy Manager to retrieve the applicable access control policies. Finally, both the DHDF and the PrimeLife XACML engines access the Request Context via the Data Reader component and retrieve the information (possibly certified by the Credential System) needed for the evaluation.

Figure 4 shows a detailed representation of the data flow of the PrimeLife XACML Engine, integrated within the PrimeLife architecture. The structure presents a strong correspondence with the standard XACML architecture, and traditional XACML components can be easily mapped onto the modules. The data flow is as follows. First, the access request is forwarded by the AC Manager to the PrimeLife PEP (step 1), which is an extended version of the standard XACML PEP. The PrimeLife PEP is responsible for managing the interaction with the AC Manager, providing functionality for supporting negotiation (i.e., request for additional data). The PrimeLife PEP component creates an XACML request and forwards it to the Context Handler (step 2). The Context Handler, as in the standard XACML architecture, invokes the PrimeLife PDP (step 3). The PrimeLife PDP is responsible for evaluating traditional XACML policies and extends the standard XACML PDP component to access the Policy Manager. The PrimeLife PDP then accesses the Policy Manager to

retrieve the XACML policies applicable to the request (step 4). To this aim, the Policy Manager provides an external interface compliant with a standard PAP component. After collecting the applicable policies, the PrimeLife PDP accesses the Context Handler to retrieve the data needed for the evaluation (step 5). If some data necessary for taking a decision are missing, the Context Handler may access different PIP extensions. These extensions include an enhanced PIP (PrimeLife PIP) that communicates with the Data Reader to access the Request Context (step 6-9). After receiving all data, the PrimeLife PDP (step 10) finally evaluates the applicable policies and takes a decision. The final decision is sent to the Context Handler (step 11) and then forwarded to the PrimeLife PEP component (step 12). Finally, the access decision is returned to the AC Manager (step 13), which combines it with the answer coming from the evaluation of data handling policies performed by the DHDF.

Based on the requirements introduced in Section 1, we present in the following the main advantages achieved by the proposed architecture with respect to the standard XACML implementation.

- *Negotiation (minimal disclosure and interactive enforcement).* The current XACML architecture does not support negotiation between the parties. An XACML access decision can assume four values (i.e., *permit*, *deny*, *indeterminate*, *not applicable*). Indeterminate is returned if no decision can be taken since some information needed to evaluate the policy is missing. For instance, an access is restricted to users of a certain country, but the nationality of the requester is unknown. If the decision returns indeterminate, according to a safe-default approach, XACML simply denies the access. Our solution enhances XACML with the ability of dialoguing with the user to communicate the fact that certain information needed for evaluating the access control policy is missing, allowing then the user to provide it and, possibly, successfully acquire access. The Decision Wrapper is responsible for the dialogue between the server and the user that consists in communicating all the conditions that have to be satisfied. At each access control request, if the policy evaluation results indeterminate, the AC Manager returns to the user the attributes that need to be evaluated for taking the access decision. The user can then decide to provide the requested attributes. Internally, the processing of the response to the user works as follows. When the PrimeLife PDP component generates an *indeterminate* response, it returns such a response to the PrimeLife PEP together with the information on which attributes are needed for a decision to be taken. This communication is enforced via a side channel provided by the PrimeLife PIP, and accessible by the PrimeLife PEP component. The use of a side channel is necessary to be able to communicate attributes, since the response format used by the standard XACML PDP engine can assume only the allowed four values and cannot represent that information. At evaluation time, the PrimeLife PIP component has the responsibility of identifying the attributes that are unknown and need to be evaluated for taking the access control decision. PrimeLife support for negotiation allows the client and the server to incrementally exchange data and request for data preserving, on one side, the principle of the *minimum disclosure* and, on the other side, supporting an incremental evaluation of the policies.

- *Credential-based restrictions.* While XACML acknowledges that properties can be presented by means of certificates, it does not provide a real support for expressing and reasoning about digital credentials in the specification and evaluation of the authorization policies. By contrast, PRIME supports requests for credentials and certified attributes in the policies. Also, PRIME can reason about credentials. For instance, PRIME supports the definition of policies requesting information (i.e., a set of attributes) to be certified by the same credential. Credential information is stored in the Request Context and is accessible by the PrimeLife XACML Engine. The XACML architecture is then enhanced to support credential-based restrictions, called *evidences* in the PRIME architecture, without requiring changes to the language schema. Rather, the XACML language is enhanced with credential specification by changing the semantics of the *issuer* attribute, originally defining the issuer of a specific attribute. In our solution, the *issuer* attribute is used as a reference to a specific evidence, stored as a separate XML file and containing the restrictions on credentials, to be evaluated on credential metadata (e.g., issuer, type). The PrimeLife XACML Engine (see Figure 4) evaluates credential-based restrictions as follows: *i)* the PrimeLife PDP component loads a policy, together with information related to the evidences; *ii)* through the PrimeLife PIP component, the PrimeLife PDP accesses information needed to evaluate the evidences of the policies and stored in the Request Context; *iii)* the PrimeLife PDP evaluates the policies and the evidence conditions on the credential metadata.

- *User-driven constraints.* While XACML is designed to manage access control only, and is focused on service side restrictions, the system illustrated in this paper takes a different approach and aims at providing the users with a solution for protecting their privacy. The PrimeLife solution, in fact, in addition to policies and mechanism to manage and evaluate traditional access control policies, provides a fully compatible and extensible solution that supports the definition, evaluation, and enforcement of privacy policies defined by the users themselves. These privacy policies (i.e., data handling policies) are defined through an ad-hoc syntax and are evaluated together with the XACML policies to provide a full-fledged privacy-aware access control system, that permits users to protect their own data.

5. SOME NOTES ON IMPLEMENTATION

The PrimeLife framework is implemented in Java, and extends and integrates the last version of the PRIME integrated prototype [10] and the SUN implementation [14] of the XACML engine, released under a BSD-like SUN license.

We focus the discussion on the mechanisms used to setup the dialog between parties, to manage credential evaluation, and to evaluate data handling policies. These functionalities represent the most important paradigm shift with respect to

the current and available XACML implementations. First, the XACML engine has been extended to support credential and dialog management, and then has been integrated with DHDF to support data handling policy evaluation.

With reference to credential and dialog management, we identified three main points of extensions of the XACML engine, producing the PrimeLife PEP, the PrimeLife PDP, the PrimeLife PIP components, respectively as Java specializations of the PEP, PDP, and PIP classes. The PrimeLife PEP supports requests for additional data (i.e., dialog), the PrimeLife PDP provides integration with the Policy Manager, and the PrimeLife PIP provides an interface with the Data Reader to access the Request Context. In addition, the Data Reader is a key component for credential management and evaluation. It represents the bridge between the Request Context and the PrimeLife PIP, and implements the functionality to access data stored in the context (including information on the credential evidences) for the XACML policy evaluation. The Data Reader also allows the PrimeLife PIP to collect and process missing attributes, thus enabling dialog management and interactive enforcement.

Methods *getSpecialFunctionResult* and *getDataWithEvidence* of class Data Reader are of particular interest. Figure 5 illustrates their codes, which for the sake of clarity has been simplified by substituting less significant parts with comments (in italics).

Method *getSpecialFunctionResult* is a recursive method responsible for the evaluation of those attributes that are derived from others (e.g., attribute *age* is derived from the date of birth and the current date). To this aim, a specific ontology of concepts (accessible by the Request Context) has been created to map derived attributes to elementary ones, and to identify the correct Java function to be applied for their evaluation. In particular, *getSpecialFunctionResult*: *i)* retrieves all the attributes needed for the evaluation (line 8), *ii)* checks if some of these attributes are derived, and, if so, recursively applies itself (lines 12-13), *iii)* identifies, via Java reflection, the Java function to be invoked for the evaluation of the derived attribute (lines 26-27).

Method *getDataWithEvidence* is responsible to access the Request Context to retrieve a list of attributes values (i.e., a pair $\langle URI, DataType \rangle$, where: *URI* is the attribute name, and *DataType* is the attribute value) that match the requested evidence (i.e., the input argument *EvidenceXML*). The method takes evidence restrictions in input and checks if they are satisfied by the data released by the requester and stored in the context. Specifically, *getDataWithEvidence*: *i)* creates an *Evidence* instance from the requested evidence *EvidenceXML* (line 7), *ii)* retrieves all the attributes and related values associated with the evidence (lines 9-14), *iii)* for each retrieved attribute checks whether it is a derived attribute (line 25), and either applies *getSpecialFunctionResult* to it (line 26) or retrieves the values from the context (line 33), *iv)* creates a *DataType* with the current attribute values (lines 29-30 and lines 37-38). Note how the definition of evidence restrictions and Request Context implementation enrich XACML with the possibility of specifying conditions to be evaluated on certified data. The certified information relevant to the evaluation process is retrieved from the Request Context component by using the evidence associated with the statement of the policy to be evaluated. The link to the evidence document is realized by means of the XACML *issuer* attribute. This permits to define conditions

```
1  public Pii getSpecialFunctionResult(
2      final URI U,
3      final Map<URI, List<Data>> DataRepository,
4      final int CurrentLevel) throws Exception {
5
6    // in case of infinite loop (watchgdog), throw an exception
7
8    List<URI> StartingArguments = MyRequestContext.getArgument(U);
9    List<Pii> ListOfArgument = new ArrayList<Pii>();
10
11   for (URI A : StartingArguments) {
12     if ( MyRequestContext.isSpecial(A) ) {
13       Pii Tmp = getSpecialFunctionResult(A, DataRepository, CurrentLevel + 1);
14       if (null != Tmp) {
15         ListOfArgument.add(new Pii(Tmp.getCategory(), Tmp.getValue()));
16       }
17     } else {
18       List<Data> ListOfData = DataRepository.get(A);
19       if ( ListOfData.size() > 0 ) {
20         Data Atom = ListOfData.get(0);
21         ListOfArgument.add( new Pii(Atom.getCategory(), Atom.getValue()) );
22       } // else store missing attribute for negotiation
23     }
24   }
25
26   Method Method2Call = MyRequestContext.getMethod(U);
27   Pii Result = (Pii)Method2Call.invoke(null, MyRequestContext, ListOfArgument);
28
29   return new Pii(U, Result.getValue());
30 }
```

(a)

```
1  public List<Map<URI, DataType>> getDataWithEvidence(
2      final String EvidenceXML,
3      final Collection<URI> Uris) {
4
5    List<Map<URI, DataType>> Result = new
   ArrayList<Map<URI, DataType>>();
6
7    Evidence evidence = StringUtils.unmarshal(EvidenceXML, Evidence.class);
8
9    // retrieve all the attributes (URIs) releated with the evidence and mark
10   // the derived ones
11
12   // store the result in (List<URISpecialCouple>) EvidenceUris
13
14   Data[][] RCResult = MyRequestContext.getEvidenceGroups(
15                                   MyRequestContext.getSubject(),
16                                   evidence,
17                                   extractURIs(EvidenceUris) );
18   for (int g = 0; g < RCResult.length; ++g) {
19     // collect attributes linked to the requested evidence and store
20     // them TmpURIValuesPair
21
22     Map<URI, DataType> URIValuesPair = new HashMap<URI, DataType>();
23
24     for (URI U : Uris) {
25       if ( MyRequestContext.isSpecial(U) ) {
26         Pii SFR = getSpecialFunctionResult(U, TmpURIValuesPair, 0);
27
28         if (null != SFR) {
29           // create an instance of the attribute and store it in
30           // URIValuesPair from SFR
31         }
32       } else {
33         List<Data> ListOfData = TmpURIValuesPair.get(U);
34
35         if (null != ListOfData) {
36           for(Data d : ListOfData) {
37             // create an instance of the attribute and store it in
38             // URIValuesPair from d
39           }
40         }
41       }
42     }
43     // if all attributes are contained within the solutions, therefore add
44     // solutions to Result
45   }
46   return Result;
47 }
```

(b)

Figure 5: Methods getSpecialFunctionResult (a) and getDataWithEvidence (b)

to be satisfied by a specific evidence. The XML fragment in Figure 6 shows an evidence restriction requiring an *X.509 identity document* released by the *Italian Public Administration*.

Concerning data handling management, DHDF has been designed to be thread-safe, to support multiple instance execution. The mechanism used to evaluate the policies relies

```
<evidences>
  <evidence>
    <issuer>ItalianPublicAdministration</issuer>
    <proofMethod>X.509</proofMethod>
    <type>identity−document</type>
  </evidence>
</evidences>
```

Figure 6: An example of evidence restriction

on a *Reverse Polish Notation* (RPN) stack. The RPN stack-based evaluation has the main advantages of being very fast and of making the evaluation process independent from policy syntax and semantics. The translation from the policy language to the RPN format is made by a specific Java class, called *DHPolicyLoader*, that interprets the syntax of the DHPs and fill in the RPN stack. In this way, it is possible to add new policy languages by specifying a new loading class, with a minimal impact on the implementation.

To conclude, the results of the data handling policy evaluation are combined and reconciled with the results produced by the evaluation of the PrimeLife XACML access control policies to produce a single decision that considers the privacy preferences of the users.

6. RELATED WORK

Previous work most directly related to ours is in the area of access control and trust management. Recent proposals have worked towards languages and models able to express access control policies, where the access decision is taken on the basis of some properties, possibly proven by presenting one or more certificates that the requesting party may have (e.g., [3, 5, 17, 18]). Winslett et al. [17] have been the first to investigate the application of credential-based access control for regulating access to a server. In [17], access control rules are expressed in a logic language and rules applicable to an access can be communicated by the server to the clients. Jajodia et al. [5] define the Flexible Authorization Framework (FAF), a logic-based framework that balances flexibility and expressiveness on one side, and easy management and performance, on the other side. FAF is based on an access control model that does not depend on any policy but is able to represent different policies and protection requirements. Bonatti and Samarati [3] provide a uniform framework for attribute-based access control specification and enforcement. Access rules are specified as logical rules, with some predicates explicitly identified. Attribute certificates are modeled as credential expressions. Besides credentials, this proposal also permits to reason about declarations (i.e., unsigned statements) and profiles of the users that a server can exploit to reach an access decision. The expressive power and the formal foundation of all these logic-based solutions permit to express complex protection requirements in a simple yet effective way. However, such proposals are not immediately suited to the Internet context considered in PrimeLife, where simplicity and easy integration with existing technologies must be ensured. Also, they are focused on access control, but do not take into account the problem of secondary use of personal information.

In the literature, several trust negotiation strategies have been also proposed (e.g., [6, 12, 18]). Trust negotiation occurs whenever credentials themselves carry some sensitive information. In such a situation, a procedure needs to be applied to establish trust through negotiation, where trust is established gradually by disclosing credentials and requests for credentials. For instance, Winslett et al. [6] present Traust, a third-party authorization service that is based on the TrustBuilder framework for trust negotiation. The Traust service provides a negotiation-based mechanism that allows qualified users to obtain the credentials necessary to access resources provided by the involved server. Some research efforts have also directly tackled the issue of supporting privacy-aware access control [2, 4, 13, 15, 16]. Notwithstanding the benefits of all these works (e.g., credential integration), none of them provides a complete and full-fledged solution for providing access control, protecting privacy of users, and regulating the use of their personal information in secondary applications.

7. CONCLUSIONS

We presented the design of a privacy-aware access control system offering compatibility with the XACML standard and support for the rich management of privacy preferences, that we have implemented within the PrimeLife project. Our solution, integrating an enhanced XACML engine with the PRIME Access Control system, can provide benefits both in the mid-term and in the long-term perspectives.

In the mid-term, the design presented in the paper will be the basis of a working system satisfying client-side and server-side privacy requirements of complex Web applications. The solution offers features that are only partially supported by other systems, both from the industry and from the academic world. Information on the design of the solution can help the development of applications making good use of the services offered by this platform.

In the long-term, it is natural to expect that XACML will be extended in order to offer an extensive support for the management of privacy requirements. The design of the future XACML extension can receive a significant benefit from the consideration of the system presented in this paper, which identifies where it is necessary to operate in order to introduce adequate support for privacy, at the same time making clear how most of the XACML architecture can be confirmed, reusing the support for extensions already present in the current XACML standard.

8. ACKNOWLEDGMENTS

This work was supported in part by the EU within the 7th Framework Programme (FP7/2007-2013) under grant agreement no. 216483 “PrimeLife”.

9. REFERENCES

[1] C.A. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati. A privacy-aware access control system. *Journal of Computer Security (JCS)*, 16(4):369–392, 2008.

[2] P. Ashley, S. Hada, G. Karjoth, and M. Schunter. E-P3P privacy policies and privacy authorization. In *Proc. of the ACM Workshop on Privacy in the Electronic Society (WPES 2002)*, Washington, DC, USA, November 2002.

[3] P. Bonatti and P. Samarati. A unified framework for regulating access and information release on the Web. *Journal of Computer Security*, 10(3):241–272, 2002.

[4] *eXtensible Access Control Markup Language (XACML) Version 2.0*, February 2005. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[5] S. Jajodia, P. Samarati, M.L. Sapino, and V. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems (TODS)*, 26(2):214–260, June 2001.

[6] A. Lee, M. Winslett, J. Basney, and V. Welch. The Traust authorization service. *ACM Transactions on Information and System Security (TISSEC)*, 11(1):1–3, February 2008.

[7] OASIS. *Security Assertion Markup Language (SAML) V1.1*, 2003. http://www.oasis-open.org/committees/security/.

[8] Object Management Group. *The CORBA Security Service Specification*. ftp://ftp.omg.org/pub/docs/ptc.

[9] PrimeLife - Privacy and Identity Management in Europe for Life . http://www.primelife.eu/.

[10] Privacy and Identity Management for Europe (PRIME). http://www.prime-project.eu.org/.

[11] P. Samarati and S. De Capitani di Vimercati. Access control: Policies, models, and mechanisms. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, LNCS 2171. Springer-Verlag, 2001.

[12] K. Seamons, M. Winslett, and T. Yu. Limiting the disclosure of access control policies during automated trust negotiation. In *Proc. of the Network and Distributed System Security Symposium (NDSS 2001)*, San Diego, CA, USA, April 2001.

[13] N. Seki and M. Kudo. Access control policy for XML. In M. Gertz and S. Jajodia, editors, *Handbook of Database Security: Applications and Trends*. Springer-Verlag, 2007.

[14] Sun's XACML Implementation. http://sunxacml.sourceforge.net/.

[15] W3C. *Platform for privacy preferences (P3P) project*, April 2002. http://www.w3.org/TR/P3P/.

[16] Web services policy framework. http://www.ibm.com/developerworks/webservices/library/specification/ws-polfram/?S_TACT=105AGX04&S_CMP=LP, March 2006.

[17] M. Winslett, N. Ching, V. Jones, and I. Slepchin. Assuring security and privacy for digital library transactions on the web: Client and server security policies. In *Proc. of the 4th International Forum on Research and Technology Advances in Digital Libraries (ADL '97)*, Washington, DC, USA, May 1997.

[18] T. Yu, M. Winslett, and K. Seamons. Supporting structured credentials and sensitive policies trough interoperable strategies for automated trust. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):1–42, February 2003.