Adding Support to XACML for Dynamic Delegation of Authority in Multiple Domains

David W Chadwick, Sassa Otenko, and Tuan Anh Nguyen

University of Kent, Computing Laboratory, Canterbury, Kent, CT2 7NF
d.w.chadwick@kent.ac.uk, o.otenko@kent.ac.uk, tn32@kent.ac.uk

Abstract. In this paper we describe how we have added support for dynamic delegation of authority that is enacted via the issuing of credentials from one user to another, to the XACML model for authorisation decision making. Initially we present the problems and requirements that such a model demands, considering that multiple domains will typically be involved. We then describe our architected solution based on the XACML conceptual and data flow models. We also present at a conceptual level the policy elements that are necessary to support this model of dynamic delegation of authority. Given that these policy elements are significantly different to those of the existing XACML policy, we propose a new conceptual entity called the Credential Validation Service (CVS), to work alongside the XACML PDP in the authorisation decision making. Finally we present an overview of our first specification of such a policy and its implementation in the corresponding CVS.

Keywords: XACML, Delegation of Authority, Credentials, Attributes, Policies, PDP.

1 Introduction

XACML is an OASIS standard for authorisation decision making. Many people are starting to experiment with it in their applications e.g. [11, 12]. Some of its benefits include: a flexible attribute based authorisation model, where access control decisions can be made based on the attributes of the subject, the action and the target; a comprehensive way of specifying conditions, so that arbitrarily complex conditions can be specified; and the support for obligations.

However, one of the current drawbacks of using XACML is that it does not support dynamic delegation of authority. A delegate is defined as "A person authorized to act as representative for another; a deputy or an agent" [1]. Delegation of authority is the act of one user with a privilege giving it to another user (a delegate), in accordance with some delegation policy. A delegation tree may thus be created, starting from the root user who has the privilege initially, to the users at the leaves of the tree who end up with the authority to assert the delegated privilege, but cannot delegate it themselves. Non leaf nodes in the tree are authorities (or administrators) with permission to delegate, but may or may not be able to assert the privilege themselves (according to the delegation policy). We differentiate between static and dynamic delegation of authority, in that static delegation of authority is when the non leaf nodes of the delegation tree are configured into software (or policy)

H. Leitold and E. Markatos (Eds.): CMS 2006, LNCS 4237, pp. 67 – 86, 2006.

prior to user access i.e. the shape of the delegation tree is known from the start, and no new non-leaf nodes can be created without reconfiguring the software (or policy). Dynamic delegation of authority is when only the root user and delegation policy are configured into the software prior to user access, and users may dynamically delegate authority to other users as and when they wish. In this case the delegation tree is created dynamically as one user delegates to another, and new leaf (and non-leaf) nodes are created spontaneously.

A responsive authorisation infrastructure that can cater for rapidly changing dynamic environments should be able to validate the privileges given to any of the users in a dynamically created delegation tree, even though the actual tree is not known when the authorisation policy is written and fed into the policy decision point (PDP). This requires the authorisation policy to be supplemented with a delegation policy that will state how the delegation tree is to be constrained. As long as a user's credential falls within the scope of the delegation tree then it is considered valid, if it falls outside the tree, and thus outside the delegation policy, it is not. The purpose of the current research was to add dynamic delegation of authority to an authorisation infrastructure that contains an XACMLv2 PDP (or in fact any PDP that bases its access control decisions on the attributes of subjects), without changing the XACMLv2 PDP or its policy[1].

We assume that privileges can be formulated as attributes and given to users. An important point to clarify at the outset is the difference between an attribute and a credential (i.e. authorisation credential). An attribute is a property of an object[2]; a credential is a *statement* or *assertion* about an attribute (in particular, a credential must state: what the attribute is, who the attribute belongs to, who says so (i.e. who is the credential issuer), and any constraints on its validity). Because attributes of an entity do not always exist as part of the entity, they are often stored or transferred as separate stand alone credentials. In this paper we are concerned with dynamic delegation of authority from one user to another by the use of credentials. One important feature of a credential is that it requires validation before the user can be attributed with the asserted property.

The rest of this paper is structured as follows. Section 2 describes the problems that need to be addressed when creating an infrastructure to support dynamic delegation of authority between multiple domains, and this leads to various requirements being placed on any proposed solution. Section 3 describes the new conceptual credential validation service (CVS) that is proposed to resolve the problems and requirements described in Section 2. Section 4 briefly describes the XACMLv2 infrastructure. Section 5 discusses how the CVS could be incorporated into the XACML infrastructure. Section 6 describes our implementation of a CVS. Section 7 concludes, and looks at possible future work in this area.

---

[1] Note that this research started whilst XACMLv2 was still under construction, when it was known that XACMLv2 would not support dynamic delegation of authority. This was one of the reasons for not proposing changes to XACMLv2. Work is currently underway to add administration and delegation to XACML v3 [18], but this is complementary to the work described here.

[2] Dictionary.com defines an attribute as "A quality or characteristic inherent in or ascribed to someone or something".

2 Problem and Requirement Statements

The underlying model used for dynamic delegation of authority in multiple domains is an enhancement of the basic XACMLv2 model (see later). In this enhanced model a user (subject) is dynamically given a set of attributes by one or more dynamically created attribute authorities (AAs) in one or more domains, and these attributes are presented (pushed) to or obtained (pulled) by the PDP as a set of credentials (usually in the form of attribute assertions digitally signed by the AAs). The PDP makes its access control decisions based on its policy, the validated set of subject attributes, the target and environmental attributes and the parameters of the user's request. Below are a set of issues that need to be addressed in such a model.

1. **Valid vs. Authentic Credentials.** The first thing to recognise is the difference between an *authentic* credential and a *valid* credential. An *authentic* credential, from the perspective of authorisation decision making, is one that has been received exactly as it was originally issued by the AA. It has not been tampered with or modified. Its digital signature, if present, is intact and validates as trustworthy meaning that the AA's signing key has not been compromised, i.e. his public key (certificate) is still valid. A *valid* credential on the other hand is one that is trusted by the PDP's policy for authorisation decision making. In order to clarify the difference, an example is the paper money issued by the makers of the game Monopoly. This money is authentic, since it has been issued by the makers of Monopoly. The money is also valid for buying houses on Mayfair in the game of Monopoly. However, the money is not valid if taken to the local supermarket.
2. **Credential validity is determined by target domain.** The above discussion leads onto the second problem that needs to be addressed in any solution, and this is that there are potentially *multiple domains* within an authorisation infrastructure. There are issuing domains, which issue credentials, and target domains that consume credentials. The PDP is part of the target domain, and as such it must use the policy of the target domain to decide whether a credential is to be trusted or not i.e. is valid or not. So the validity of an authorisation credential is ultimately determined by the (writer of the) PDP policy. A valid credential is a credential that is trusted by the consumer of the credential.
3. **Multiple trusted credential issuers.** In any system of any significant size, there will be multiple credential issuers. Some of these will be trusted by the target domain, others will not be. Thus the system must be capable of differentiating between trusted and untrusted issuers, and of dynamically obtaining this information from somewhere. (In point 4 below we propose to use roots of trust.) Different target domains in the same system may trust different issuers, and therefore the PDPs must be capable of being flexibly configured via their policies to say which issuers are trusted and which are not. For example, in the physical world of shopping with credit cards, there are several issuers such as Amex and Visa. Some shopkeepers accept (trust) both issuers, others only trust one of them. It is their (the target domain's) decision which card issuers to trust.
4. **Identifying roots of trust.** Point 3 above leads us to conclude that the PDP must be configured, in an out of band trusted way, with at least one (authorization) root

of trust and it is from this root (or roots) of trust that all credentials must be validated in order to be trusted. A root of trust must be a single entity identified directly or indirectly by its public key[3], since this key will be used to validate the signed credentials that are received. Note that it is not possible to refer to a *root of trust* through its set of assigned attributes, e.g. anyone with a project manager attribute and company X attribute, since these attributes may identify several candidate roots, and may be issued by several attribute authorities, in which case it wont be known who to trust. This implies that a higher authority is the real root of trust, the one who issues the set of attributes that can be trusted.

5. **The role of the Issuer's policy.** Most issuers will have an Issuing Policy, to say who is allowed to issue which credentials to which users, and what constraints are placed on their use. This policy will include the delegation policy of the issuer. Consequently there will be constraints on which credentials are deemed to be valid within and without the issuing domain. However, the target domain may choose to ignore these constraints and trust (treat as valid) credentials which the issuer deems to be invalid. A well known example in the physical world concerns supermarkets who issue their own discount coupons. These coupons state quite clearly that they are only valid for use in supermarkets owned by the issuer. However, it is often the case that a different brand of supermarket will accept these discount coupons as a way of enticing the other supermarkets' customers to come and shop in their own supermarket. Thus the PDP must have a way of either conforming to or overriding the issuer's policy. If a target domain chooses to ignore the issuer's policy, then it is liable for any losses incurred by this. The issuer cannot be held responsible for targets that ignore its Issuing Policy.
6. **Obtaining the Issuing Policy.** In a multi-domain system, the target domain may not be aware of the issuing domain's Issuing Policy, unless it is explicitly placed into the issued credentials. If the complete Issuing Policy is not explicitly placed in the issued credentials, but the target domain still wishes to enforce it and only treat as valid those credentials that the issuer says are valid, then the target's PDP will need to infer or be configured with the issuer's Issuing Policy. For example, in SPKI [7], a credential is marked as being infinity delegatable or not, and does not contain any other details of the Issuing Policy, such as who is entitled to be delegated the privilege. Thus unless a delegatable credential explicitly contains restrictions, or out of band means are used to transfer them, the target PDP will infer than anyone is entitled to be delegated this credential.
7. **Pulling credentials.** The PDP may not have all the credentials it needs in order to validate the credential(s) presented by the user, e.g. if only the leaf credential in a delegation tree is presented, but none of the non-leaf credentials are presented. In the most extreme case the user may not present any credentials at all. For example, when a user logs into a portal and the portal displays only the services this user is allowed to see, the portal has, unknown to the user, retrieved the user's credential(s) from a repository in order to determine which services to

[3] When an X.509 conformant PKI is used which already has its own configured CA root public keys, the globally unique name of the subject in the PKI certificate can be used to refer to the authorization root of trust, instead of the public key in the certificate, in which case the subject will be trusted regardless of which public/private key pair it is currently using.

display. There is thus a strong requirement for the PDP (or a component of it) to be able to pull the user's credentials before making the access control decision.

8. **Discovering credential locations.** The user's credentials may be stored and/or issued in a variety of places, for example, each AA may store the attributes or the credentials it issues in its own repository. One could always mandate that the user collects together the credentials he wants to use, before attempting to gain access to a resource e.g. as in the VOMS model [13]. This model has its merits in some cases, but it is not always very user friendly. In fact, in some cases, the user may not be aware what credentials have actually been issued to him – he might only know what services he is allowed to access, as in the portal example given above. In the general case there is no absolute requirement for the user to know what credentials have been issued to him or where they are stored. Thus the PDP must be capable of contacting different repositories/AAs in order to pull the user's credentials prior to making its access control decision.
9. **Multiple user identities.** If the user is known by different identities to the different AAs, then there must be a way for the user to use these mixed credentials in the same session. The GridShib project currently uses a mapping table to convert between X.509 PKI identities and Shibboleth identity provider identities [14]. But a more flexible approach is needed, in which the user may determine which set of credentials are to be used in a given session and the PDP can prove the user's right to assert each one. We propose one solution to this in [20].
10. **Multiple credential formats.** Following on from above, the user's credentials may be stored in different formats in the different repositories, and presented to the PDP in different ways, e.g. as signed SAML assertions [2], as X.509 attribute certificates [3], as Shibboleth encoded attributes [4] etc. The PDP (or a component of it) therefore needs to be able to decode and handle credentials in different formats.
11. **Hierarchies of attributes.** The attributes may form some sort of hierarchy, for example in accordance with the NIST $RBAC_1$ specification [5], in which the superior attributes (or roles) inherit the privileges of the subordinate roles. The PDP needs to be aware of this hierarchy when validating the credentials. For example, if a superior role holder delegates a subordinate role to another user, then the PDP needs to know if this delegation is valid or not, given that the attributes are different. Furthermore some of the attributes known to the PDP won't form a hierarchy. Therefore the PDP needs to be able to cater for multiple disjoint attribute hierarchies.
12. **Constraining credential validity.** Only part of an authentic credential might be valid in a target domain. For example, a credential might contain multiple attributes but the target domain only trusts the issuer to issuer a subset of the enclosed attributes.
13. **Known and unknown attributes.** As federations between organisations become more common, and dynamic VOs become more feasible, managers will realise the need to define a common set of attributes that can be understood between domains. The US academic community realised this some time ago, and this led to the definition of EDU person [6], which is a collection of standard attribute types. However, once organisations start to issue standard attributes, a PDP will

need to be able to differentiate between which standard attributes are valid (trusted) and which are not. For example, suppose most organisations in the world issue a standard Project Manager attribute to their project managers. In a VO between organisations A and B, the PDP in organisation B might only want to trust the Project Manager attributes issued by itself, and not those issued by organisation A (or by C or D or any other organisation). Or alternatively it might wish to downgrade those issued by organisation A and treat them as being equivalent to a guest user attribute. Or it might decide to trust the project managers from A as being equal to its own project managers. The PDP's policy needs to flexible enough to cater for all these requirements, including the ability to perform attribute mappings.

3 Architecting a Solution

Given the problem statements and various requirements from above, one can see that some new functional requirements have been placed on the PDP. Consequently, we propose a new conceptual component called a Credential Validation Service (CVS), whose purpose is to perform the new functionality. In essence the purpose of the CVS is to validate a set of credentials for a subject, issued by multiple dynamic attribute authorities from different domains, according to the local policy, and return a set of valid attributes. How this conceptual component is merged into the XACML infrastructure will be described later. The rationale for making the CVS a separate component from the XACML PDP are several. Firstly, its purpose is to perform a distinct function from the PDP. The purpose of the PDP is to answer the question "given this access control policy, and this subject (with this set of valid attributes), does it have the right to perform this action (with this set of attributes) on this target (with this set of attributes)" to which the answer is essentially a Boolean, Yes or No[4]. The purpose of the CVS on the other hand is to perform the following "given this credential validation policy, and this set of (possibly delegated) credentials, please return the set of valid attributes for this entity" to which the answer will be a subset of the attributes in the presented credentials. Secondly, the XACML language is incapable of specifying credential chains. This is because subjects and attribute issuers are identified differently in the language, hence it is not possible to chain delegated credentials together.

When architecting a solution there are several things we need to do. Firstly we need a trust model that will tell the CVS which credential issuers and policy issuers to trust. Secondly we need to define a credential validation policy that will control the trust evaluation of the credentials. Finally we need to define the functional components that comprise the CVS.

3.1 The Trust Model

The CVS needs to be provided with a trusted credential validation policy. We assume that the credential validation policy will be provided by the Policy Administration

[4] XACML also supports other answers: indeterminate (meaning an error) and not applicable (meaning no applicable policy), but these are other forms of No.

Point (PAP), which is the conceptual entity from the XACML specification that is responsible for creating policies. If there is a trusted communications channel between the PAP and the CVS, then the policy can be provided to the CVS through this channel. If the channel is not trusted, or the policy is stored in an intermediate repository, then the policy should be digitally signed by a trusted policy author, and the CVS configured with the public key (or distinguished name if X.509 certificates are being used) of the policy author. In addition, if the PAP or repository, has several different policies available to it, to be used at different times, then the CVS needs to be told which policy to use. In this way the CVS can be assured of being configured with the correct credential validation policy. All other information about which sub policies, credential issuers and their respective policies to trust can be written into this master credential validation policy by the policy author.

In a distributed environment we will have many issuing authorities, each with their own issuing policies provided by their own PAPs. If the policy author decides that his CVS will abide by these issuing policies there needs to be a way of securely obtaining them. Possible ways are that the CVS could be given read access to the remote PAPs, or the remote issuing authorities could be given write access to the local PAP, or the policies could be bound with their issued credentials and obtained dynamically during credential evaluation. Whichever way is used, the issuing policies should be digitally signed by their respective issuers so that the CVS can evaluate their authenticity.

The policy author may decide to completely ignore all the issuer's policies (see section 2 point 5), or to use them in combination with his own credential validation (CV) policy, or to use them in place of his own policy. Thus this information (or policy combining rule) needs to be conveyed as part of the CV policy.

3.2 The Credential Validation Policy

The CVS's policy needs to comprise the following components:

- a list of trusted credential issuers. These are the issuers in the local and remote domains who are trusted to issue credentials that are valid in the local domain. They are the roots of trust. This list is needed so that the signatures on their credentials and policies can be validated. Therefore the list could contain the raw public keys of the issuers or it could refer to them by their X.500 distinguished names or their X.509 public key certificates.

- the hierarchical relationships of the various sets of attributes. Some attributes, such as roles, form a natural hierarchy. Other attributes, such as file permissions might also form one e.g. *all* permissions is superior to *read, write* and *delete*; and *write* is superior to *append* and *delete*. When an attribute holder delegates a subordinate attribute to another entity, the credential validation service needs to understand the hierarchical relationship and whether the delegation is valid or not.

- a description (schema) of the valid delegation trees. This delegation policy component describes how the CVS can determine if a chain of delegated credentials and/or policies falls within a trusted tree or not. This is a rather complex policy component, and there are various ways of describing delegation trees [3, 9] with no widely accepted standard way. The essential elements should specify who is allowed to be in the tree (both as an issuer and/or a subject), what constraints apply, and what properties (attributes) they can validly have (assert) and delegate.

- a linking of trusted issuers to delegation trees. This is not necessarily a one to one mapping. Several trusted issuers may be at the root of the same delegation tree, or one issuer may be at the root of several delegation trees.

- the acceptable validity constraints of the various credentials (e.g. time constraints or target constraints). Consider for example time constraints. An issuer gives each issued credential a validity period, which may range from fairly short (e.g. minutes) to very long (e.g. several years). The primary reason for issuing short lived certificates (for other than intrinsically short lived permissions) is so that they do not need to be revoked, and therefore the relying party does not need to consult revocation lists, white lists, or OCSP servers etc. In the case of relatively long lived credentials, the policy author may have his own opinion about which credentials to trust, from a chronological perspective, and therefore may wish to place his own additional time constraints on remotely issued credentials. For example, a plumber may have a "certified plumber" credential, which is valid for 10 years from the date of issue. He may be required to pass a competence test every ten years to prove that he is conversant with the latest technology developments and quality standards before the credential is renewed. However, in the target domain, the policy author may decide that he does not want to accept anyone with a credential that is newer than one year old, due to insufficient experience on the job, or is more than 8 years old, due to doubts about competencies with the latest technologies. Consequently the CVS must be told how to intersect the validity constraints on the credential with those in the author's policy.

- finally, we need a disjunctive/conjunctive directive, to say whether for each trusted issuer and delegation tree, only the issuer's issuing and delegation policy should take effect, or only the author's policy should take effect, or whether both should take effect and valid credentials must conform to both policies.

Note that when delegation of authority is not being supported, the above policy can still be used in simplified form where a delegation tree reduces to a root node that describes a set of subjects. In this case the CV policy now controls which trusted issuers are allowed to assign which attributes to which groups of subjects, along with the various constraints and disjunctive/conjunctive directive.

XACMLv2 [8] is not a suitable instrument to express Credential Validation Policies but neither is the current working draft of XACMLv3 [18]. An important requirement for multi-domain dynamic delegation is the ability to accept only part of an asserted credential. This means that the policy should be expressive enough to specify what is the maximum acceptable set of attributes that can be issued by one Issuer to a Subject, and the evaluation mechanism must be able to compute the intersection of this with those that the Subject's credential asserts. The approaches used by XACML can only state that the asserted set of attributes or policies is fully accepted, or fully rejected. In [18] the delegation is deemed to be valid if the issuer of the delegated policy could have performed the request that the policy grants to the delegatee. We think this is a serious deficiency, which lies at the core of the XACML policy evaluation process.

We think it is a limitation on an independent issuing domain to have to take into account all the policies that the validating domain supports, so that only fully acceptable sets of credentials or policies can be issued to its subjects. Our model is based on full independence of the issuing domain from the validating domain. So in

general it is impossible for a validating domain to fully accept an arbitrary set of credentials since the issuing and validating policies will not match. It is not possible for the issuing domain to tell in advance in what context a subject's credentials will be used (unless new credentials are issued every time a subject requests access to a resource) so it is not possible to tell in advance what validation policy will be applied to them.

Having identified this problem, we propose a solution that uses a non-XACML based credential validation policy first, and an XACML policy next, with the validated delegated attributes.

3.2.1 Formal Credential Validation Policy

We define a Credential Validation Policy as an unordered set of tuples <S, I, C, E>, where S is a set of Subjects to whom any Issuer from set I can assign at most a set of Credentials C, but only if any of the conditions in set E holds true:

$$CVP = \{<S, I, C, E>\}$$

We define the Credential Validation process as a process of obtaining a subset of valid credentials V, given an asserted set of credentials c, issued by issuer i to the subject s, if condition e holds true at the time of evaluation:

$$V = \{ c \cap C \mid c \cap C \neq \emptyset, s \subseteq S, i \subseteq I, e \subseteq E, <S, I, C, E> \subseteq CVP \}$$

Note that in XACML the only possible evaluation of a Credential Validation process is:

$$V = \{ c \mid c \subseteq C, s \subseteq S, i \subseteq I, e \subseteq E, <S, I, C, E> \subseteq CVP \}$$

Further, we define a dynamic delegation process as a process of obtaining a set R of Credential Validation rules for intermediate issuers, i.e. the issuers on the path from the policy writer to the end user, where the intermediate issuer s is issued a set of Credentials c by a higher level issuer i, subject to condition e and a constraint on subject domain d:

$$R_s = \{ <d \cap S \setminus s, s, c \cap C, e> \mid c \cap C \neq \emptyset, s \subseteq S, i \subseteq I, e \subseteq E,$$

$$<S, I, C, E> \subseteq CVP \cup R_i \}$$

Thus the issuer i can allow the issuer s to delegate a subset of his own privileges to a subset of his own set of subjects, subject to the condition e being stricter than that imposed on i.

Note the recursive nature of the process - the tuple <S, I, C, E> must belong to the CVP or to the set of valid rules for issuer i. Note also that loops in the delegation are prohibited by excluding the holder of the rule from the set of possible subjects.

XACML currently lacks the expressiveness for deriving new Credential Validation rules given the set of existing rules and valid credentials.

3.3 The CVS Functional Components

Figure 1 illustrates the architecture of the CVS function and the general flow of information and sequence of events. First of all the service is initialised by giving it

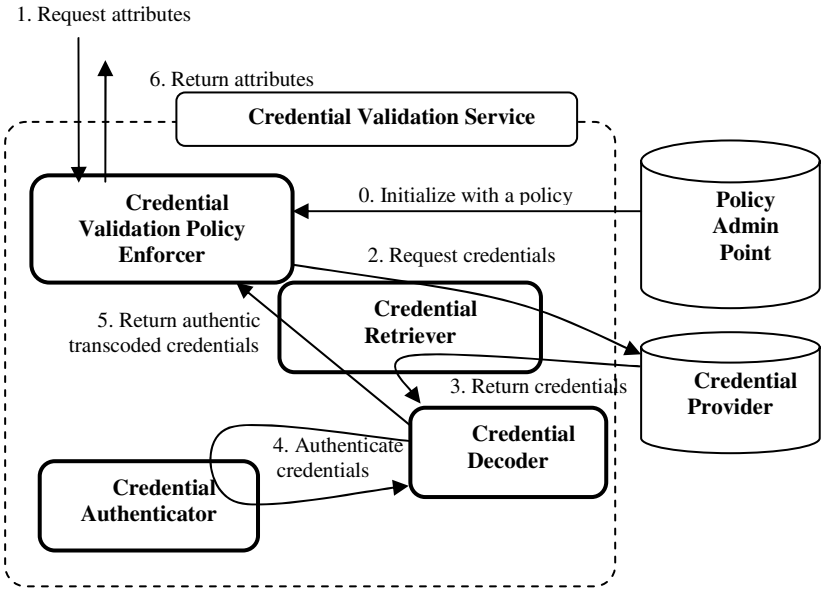

Fig. 1. Data Flow Diagram for Credential Validation Service Architecture

the credential validation policy (step 0). Now the CVS can be queried for the valid attributes of an entity (step 1). Between the request for attributes and returning them (steps 1 and 6) the following events may occur a number of times, as necessary i.e. the CVS is capable of recursively calling itself as it determines the path in a delegation tree from a given node to a root of trust. The Policy Enforcer requests credentials from a Credential Provider (step 2). When operating in credential pull mode, the credentials are dynamically pulled from one or more remote credential providers (these could be AA servers, LDAP repositories etc.). The actual attribute request protocol (e.g. SAML or LDAP) is handled by a Credential Retriever module. When operating in credential push mode, the CVS client stores the already obtained credentials in a local credential provider repository and pushes the repository to the CVS, so that the CVS can operate in logically the same way for both push and pull modes. After credential retrieval, the Credential Retriever module passes the credentials to a decoding module (step 3). From here they undergo the first stage of validation – credential authentication (step 4). Because only the Credential Decoder is aware of the actual format of the credentials, it has to be responsible for authenticating the credentials using an appropriate Credential Authenticator module. Consequently, both the Credential Decoder and Credential Authenticator modules are encoding specific modules. For example, if the credentials are digitally signed X.509 attribute certificates, the Credential Authenticator uses the configured X.509 PKI to validate the signatures. If the credentials are XML signed SAML attribute assertions, then the Credential Authenticator uses the public key in the SAML assertion to

validate the signature. The Credential Decoder subsequently discards all unauthentic credentials – these are ones whose digital signatures are invalid. Authentic credentials are decoded and transformed into an implementation specific local format that the Policy Enforcer is able to handle (step 5).

The task of the Policy Enforcer is to decide if each authentic credential is valid (i.e. trusted) or not. It does this by referring to its Credential Validation policy to see if the credential has been issued by a root of trust or not. If it has, it is valid. If it has not, the Policy Enforcer has to work its way up the delegation tree from the current credential to its issuer, and from there to its issuer, recursively, until a root of trust is located, or no further issuers can be found (in which case the credential is not trusted and is discarded). Consequently steps 2-5 are recursively repeated until closure is reached. Remember that in the general case there are multiple credential providers, who each may have their own Issuing Policies, which may be adhered to or ignored by the Policy Enforcer according to the CV policy. There are also issues of height first or breadth first upwards tree walking, or top-down vs. bottom-up tree walking. These are primarily implementation rather than conceptual issues, as they effect performance and quality of service, and so we will address them further in Section 6 where we describe our implementation of a CVS.

The proposed architecture makes sure that the CVS can:

- Retrieve credentials from a variety of physical resources
- Decode the credentials from a variety of encoding formats
- Authenticate and perform integrity checks specific to the credential encoding format

All this is necessary because realistically there is no way that all of these will fully match between truly independent issuing domains.

4 The XACML Model

Figure 2 shows the overall conceptual set of interactions, as described in XACMLv2 [8]. The PDP is initially loaded with the XACML policy prior to any user's requests being received (step 1). The user's access request is intercepted by the PEP (step 2), is authenticated, and any pushed credentials are validated and the attributes extracted (note that this is not within the scope of the XACML standard). The request and user attributes (in local format) are forwarded to the context handler (step 3), which may ask the PIP for additional attributes (steps 6 to 8) before passing the request to the PDP (step 4). If the PDP determines from the policy that additional attributes are still needed, it may ask the context handler for them (step 5). Optionally the context handler may also forward resource content (step 9) along with the additional attributes (step 10) to the PDP. The PDP makes a decision and returns it via the context handler (step 11) to the PEP (step 12). If the decision contains optional obligations they will be enforced by the obligations service (step 12).

As can be seen from Figure 2, XACMLv2 currently has nothing to say about credentials or how they are validated.

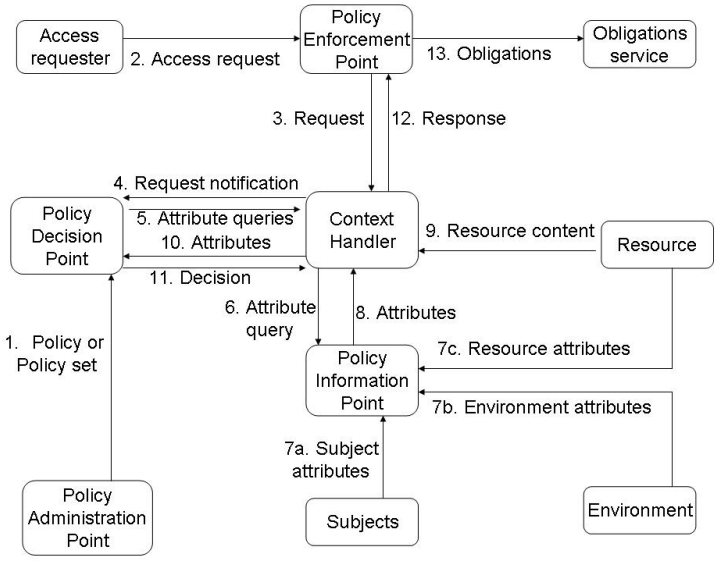

Fig. 2. Data Flow Diagram for XACML Architecture

5 Incorporating the CVS into XACML

Figure 3 shows the ways in which the CVS could be incorporated into the XACML model. The CVS could be an additional component called by either the PEP (step 101) or the context handler (step 103), or it could completely replace the PIP (step 6)

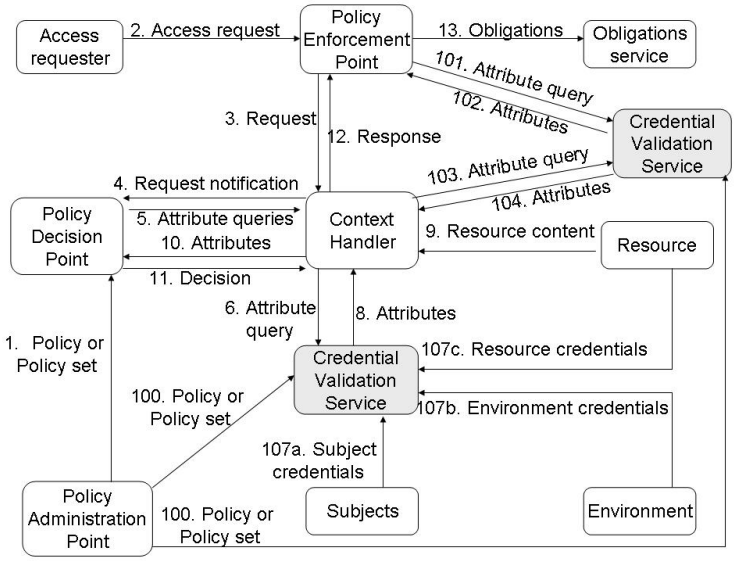

Fig. 3. Incorporating the CVS into XACML

in which case the Subject would now send credentials to the PIP/CVS rather than attributes (step 107a).

The advantages of having the CVS called by the PEP, is that existing XACMLv2 implementations do not need to change. The PEP optionally passes a bag of credentials to the CVS (push mode), the CVS fetches all or more credentials as needed (pull mode), and returns a set of valid subject attributes to the PEP, which it can pass to the existing XACML context handler. The disadvantage of this model is that each application will need to be modified in order to utilise the CVS, since the PEP is an application dependent component. Note that this model, when operating in push mode only, with no credential retrievals, is similar to that being proposed by the WS-Trust specification, in which the Security Token Service (STS) operates as a token validation service [10]. However, the STS has no equivalent functionality of the CVS operating in credential pull mode.

The advantage of having the CVS called by the context handler is that existing applications i.e. the PEP, may not need to change. The only change that is needed is to the context handler component of XACML implementations. Depending upon its interface, PEPs may not need to change at all. Support for multiple autonomous domains that each support delegation of authority can be added to applications without the application logic needing to change. Only a new credential validation policy is needed. Credentials that were previously invalid (because they had been delegated) would now become valid, once the appropriate policy is added to the PAP.

The advantage of replacing the PIP by the CVS, is that we have the opportunity of using digitally signed credentials for constructing target attributes and environmental attributes as well as subject attributes. For example, time may be obtained from a secure time stamping authority as a digitally signed credential (step 107b), and validated according to the CV policy. This is our favoured approach.

The disadvantage of the last two approaches is that incorporating the CVS inside the policy evaluator introduces transforms to the request context that are invisible to the PEP. At the current time we do not know which approach will eventually be favored.

Note that the integration scenarios do not affect the implementation of the CVS, which is explained in the next section.

6 Implementing the CVS

There are a number of challenges involved in building a fully functional CVS that is flexible enough to support the multiple requirements outlined in section 2. Firstly we need to fully specify the Credential Validation Policy, including the rules for constructing delegation trees. Then we have to engineer the policy enforcer with an appropriate algorithm that can efficiently navigate the delegation tree and determine whether a subject's credentials are valid or not.

6.1 Credential Validation Policy

We have implemented our CV policy in XML, according to our own specified DTD/schema, shown in Appendix 1. Most components of the policy are relatively

straightforward to define, apart from the delegation tree. We have specified the list of trusted credential issuers by using their distinguished names (DNs) from their X.509 public key certificates. We chose to use distinguished names rather than public keys for two reasons. Firstly, they are easier for policy writers to understand and handle, and secondly it makes the policy independent of the current key pair in use by a trusted issuer.

The attribute hierarchies are specified by listing superior-subordinate attribute value pairs. There can be multiple independent roots, and attribute values can be independent of any hierarchy if so wished.

In our first implementation, we have specified a delegation tree as a name space (domain) and a delegation depth. Anyone in the domain who is given a credential may delegate it to anyone else in the same domain, who in turn may delegate it to anyone else in the same domain until the delegation depth is reached. We currently use X.500/LDAP distinguished names to define the domains. This format allows the policy writer to define a domain using included and excluded subtrees and so construct any arbitrary LDAP subtree in which the delegates must belong. This name form is already used in the PKI world, for example in the name constraints X.509 extension [3]. Furthermore since we refer to the credential subjects and issuers by their LDAP DNs, it was natural to constrain who could be in a delegation tree by referring to them by their DNs. Another constraint that we place on a delegation tree is that the same attribute (or its subordinate) must be propagated down the tree, and new unrelated attributes cannot be introduced in the middle of a delegation tree. We recognise that a more flexible approach will be to define delegation trees by referring to the attributes of the members rather than their distinguished names, as for example is used by Bandmann et al [9]. Their delegation tree model allows a policy writer to specify delegation trees such as "anyone with a head of department attribute may delegate a project manager attribute to any member of staff in the department". This is a future planned enhancement to our work.

Finally, the policy links the trusted issuers to the delegation domains and the attributes that each issuer is trusted to issue, along with any additional time/validity constraints that are placed on the issued credentials. (The constraints have not been shown in the schema.)

In our current implementation we do not pass the full Issuing Policy along with the issued credential, we only pass the tree *depth* integer. Therefore the CVS does not know what the issuer's intended name space is (we assume that the credential issuing software will enforce the Issuing Policy on behalf of the relying party). The CV policy writer is free to specify his own (more restrictive) name space for the delegation tree, or to specify no name space restrictions and allow a credential holder to effectively delegate to anyone. The only way to (partially) enforce the Issuing Policy in our current implementation is to repeat the issuer's name space in the delegation tree of the CV policy, and to assume that no further restrictions are placed by the issuer on any particular delegate. A future planned enhancement is to carry the Issuing Policy in each issued credential, and to allow the CV policy writer to enforce it, or overwrite it with his own policy, or force conformance to both. In this way a more sophisticated delegation tree can be adhered to.

6.2 Delegation Tree Navigation

Given a subject's credential, the CVS needs to create a path between it and a root of trust, or if no path can be found, conclude that a credential cannot be trusted. There are two alternative conceptual ways of creating this path, either top down, also known as backwards [3, 17] (i.e. start at a root of trust and work down the delegation tree to all the leaves until the subject's credentials are found) or bottom up, also known as forwards (i.e. start with the subject's credential and work up the delegation tree until you arrive at its root). Neither approach is without its difficulties. Either way can fail if the credentials are not held consistently – either with the subject or the issuer. As Li et al point out [17], building an authorisation credential chain is more difficult than building an X.509 public key certificate chain, because in the latter one merely has to follow the subject/issuer chain in a tree, whereas in the former, a directed graph rather than a tree will be encountered. Graphs may arise for example when a superior delegates some privileges in a single credential that have been derived from two of more credentials that he possesses, or when attribute mappings occur between different authorities. Even for the simpler PKI certificate chains, there is no best direction for validating them. SPKI uses the forwards chaining approach [15]. As Elley et al describe in [16], in the X.509 model it all depends upon the PKI trust model and the number of policy related certificate extensions that are present to aid in filtering out untrusted certificates. Given that our delegation tree is more similar to a PKI tree, and that we do not have the policy controls to filter the top/down (backwards) approach, and furthermore, we support multiple roots of trust so in general would not know where to start, then the top down method is not appropriate.

There are two ways of performing bottom up (forwards) validation, either height first in which the immediately superior credential only is obtained, recursively until the root is reached, or breadth first in which all the credentials of the immediate superior are obtained, and then all the credentials of their issuers are obtained recursively until the root or roots are reached. The latter approach may seem counter-intuitive, and certainly is not sensible to perform in real time in a large scale system, however a variant of it may be necessary in certain cases, i.e. when two or more superior credentials have been used to create a subordinate one, or when a superior possess multiple identical credentials issued by different authorities. Furthermore, given that in our federation model described in section 2 we allow a user to simply authenticate to a gateway and for the system to determine what the user is authorised to do (the credential pull model), the first step of the credential validation process is to fetch all the credentials of the user. This is performed by the Credential Retriever in Figure 1. Thus if the CVS recursively calls itself, the breadth first approach would be the default tree walking method. Thus we have to add a tree walking directive to the credential validation method, which can be set to breadth first for the initial call to the CVS, and then to height first for subsequent recursive calls that the CVS makes to itself.

In order to efficiently solve the problem of finding credentials, we add a pointer in each issued credential that points to the location of the issuer's credential(s) which are superior to this one in the delegation tree. This pointer is similar to the AuthorityInformationAccess extension defined in [19]. Although this pointer is not

essential in limited systems that have a way of locating all the credential repositories, in the general case it is needed.

In the case of relatively long lived credentials, we envisage that a background task could be run when the system is idle, that works its way down the delegation trees from the roots, in a breadth first search for credentials, validates them against the CV policy, and caches the valid attributes for each user for a configuration period of time that is approximately equal to the revocation period. Then when a user attempts to access a resource, the CVS will be able to give much faster responses because the high level branches of the delegation tree will have already been validated.

7 Conclusions and Future Work

Providing XACML with support for dynamic delegation of authority that is enacted via the issuing of credentials from one user to another, is a non-trivial task to model and engineer. In this paper we have presented the problems and requirements that such a model demands, and have architected a solution based on the XACML conceptual and data flow models. We have also presented at a conceptual level the policy elements that are necessary to support this model of dynamic delegation of authority. Given that these policy elements are significantly different to those of the existing XACMLv2 policy, and that the functionality required to evaluate this policy is significantly different to that of the existing XACML PDP, we have proposed a new conceptual entity called the Credential Validation Service, to work alongside the PDP in the authorisation decision making. The advantages of this approach are several. Firstly the XACML policy and PDP do not need to change, and support for dynamic delegation of authority can be phased in gradually. The exact syntax and semantics of the new policy elements can be standardised with time, based on implementation experience and user requirements. We have presented our first attempt at defining and implementing such a policy, and now have an efficient implementation that supports dynamic delegation of authority. A live demonstration is available at http://issrg-beta.cs.kent.ac.uk:8080/dis.html.

Future work will look at supporting more sophisticated delegation trees and schema, and enforcing (or ignoring) Issuing Policies in target domains by passing the full policy embedded in the issued credentials. We also plan to incorporate additional policy elements in the delegation trees, such as attribute mappings of the kind described in [18].

Acknowledgments. We would like to thank the UK JISC for supporting this work under the research project entitled “Dynamic Virtual Organisations in e-Science Education (DyVOSE)”.

References

1. See http://dictionary.reference.com/search?q=delegate
2. OASIS. “Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0”, 15 January 2005

3. ISO 9594-8/ITU-T Rec. X.509 (2001) The Directory: Public-key and attribute certificate frameworks
4. Scot Cantor. "Shibboleth Architecture, Protocols and Profiles, Working Draft 02, 22 September 2004, see http://shibboleth.internet2.edu/
5. David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn And Ramaswamy Chandramouli. "Proposed NIST Standard for Role-Based Access Control". ACM Transactions on Information and System Security, Vol. 4, No. 3, August 2001, Pages 224–274
6. Internet2 Middleware Architecture Committee for Education, Directory Working Group (MACE-Dir) "EduPerson Specification (200312)", December 2003. Available from http://www.nmi-edit.org/eduPerson/internet2-mace-dir-eduperson-200312.html
7. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, T. Ylonen. "SPKI Certificate Theory". RFC 2693, Sept 1999.
8. "OASIS eXtensible Access Control Markup Language (XACML)" v2.0, 6 Dec 2004, available from http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
9. O. Bandmann, M. Dam, and B. Sadighi Firozabadi."Constrained delegation". In Proceedings of the IEEE Symposium on Research in Security and Privacy, pages131-140, Oakland, CA, May 2002. IEEE Computer Society Press.
10. Paul Madsen. "WS-Trust: Interoperable Security for Web Services". June 2003. Available from http://webservices.xml.com/pub/a/ws/2003/06/24/ws-trust.html
11. Markus Lorch , Seth Proctor , Rebekah Lepro , Dennis Kafura , Sumit Shah. "First experiences using XACML for access control in distributed systems". Proceedings of the 2003 ACM workshop on XML security, October 31-31, 2003, Fairfax, Virginia
12. Wolfgang Hommel. "Using XACML for Privacy Control in SAML-based Identity Federations". In 9th IFIP TC-6 TC-11 Conference on Communications and Multimedia Security (CMS 2005), Springer, Salzburg, Austria, September 2005
13. Alfieri R, et al. VOMS: an authorization system for virtual organizations, 1st European across grids conference, Santiago de Compostela. 13-14 February 2003. Available from: http://grid-auth.infn.it/docs/VOMS-Santiago.pdf
14. Tom Barton, Jim Basney, Tim Freeman, Tom Scavo, Frank Siebenlist, Von Welch, Rachana Ananthakrishnan, Bill Baker, Kate Keahey. "Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy". To be presented at NIST PKI Workshop, April 2006.
15. Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, Ronald L. Rivest. "Certificate chain discovery in SPKI/SDSI". Journal of Computer Security, Issue: Volume 9, Number 4 / 2001, Pages: 285 - 322
16. Y. Elley, A. Anderson, S. Hanna, S. Mullan, R. Perlman and S. Proctor, "Building certificate paths: Forward vs. reverse". *Proceedings of the 2001 Network and Distributed System Security Symposium (NDSS'01)*, Internet Society, February 2001, pp. 153–160.
17. Ninghui Li, William H. Winsborough, John C. Mitchell. "Distributed credential chain discovery in trust management".Journal of Computer Security 11 (2003) pp 35–86
18. XACML v3.0 administration policy Working Draft 05 December 2005. http://www.oasis-open.org/committees/documents.php?wg abbrev=xacml.
19. Housley, R., Ford, W., Polk, W., and Solo, D., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 3280, April 2002
20. David Chadwick. "Authorisation using Attributes from Multiple Authorities" in Proceedings of WET-ICE 2006, June 2006, Manchester, UK

Appendix 1: CVS Policy Schema

```
<?xml version="1.0" >
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:permis="http://sec.cs.kent.ac.uk/permis" elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="CVSPolicy" type="permis:CVSPolicyType"/>
   <xs:complexType name="CVSPolicyType" >
            <xs:sequence>
                     <xs:element name="TrustedIssuers" type="permis:TrustedIssuersType" />
                     <xs:element name="AttributeHierarchies"
type="permis:AttributeHierarchiesType" />
                      <xs:element name="Domains" type="permis:DomainsType"/>
                     <xs:element name="AttributeAssignments"
type="permis:AttributeAssignmentsType" />
             </xs:sequence>
            <xs:attribute name="CVSPolicyID" use="required" type="xs:anyURI"/>
   </xs:complexType>
<!-- -->
<xs:complexType name="TrustedIssuersType">
            <xs:sequence>
            <xs:element name="TrustedIssuer" maxOccurs="unbounded"
type="permis:TrustedIssuerType"/>
            </xs:sequence>
</xs:complexType>
<!-- -->
<xs:complexType name="TrustedIssuerType">
            <xs:attribute name="TrustedIssuer" use="required" type="xs:anyURI"/>
  <!-- Only LDAP and HTTP URLs are currently allowed for issuers -->
            <xs:attribute name="TID" use="required" type="xs:ID"/
</xs:complexType>
<!-- -->
<xs:complexType name="AttributeHierachiesType">
            <xs:sequence>
            <xs:element name="AttributeHierarchy" maxOccurs="unbounded"
type="permis:AttributeHierarchyType"  />
            </xs:sequence>
</xs:complexType>
<!-- -->
<xs:complexType name="AttributeHierachyType">
            <xs:sequence>
            <xs:element name="Superior" type="permis:SuperiorValueType"
maxOccurs="unbounded" >
            <xs:sequence>
            <xs:attribute name="AttributeOID" use="required" type="xs:anyURI"/
            <!-- Must be encoded according to SAML LDAP Profile e.g. urn:oid:1.2.3.4 -->
            <xs:attribute name="FriendlyName" use="required" type="xs:ID"/
</xs:complexType>
<!-- -->
 <xs:complexType name="SuperiorValueType">
            <xs:sequence>
```

```
            <xs:element name="Subordinate" type="permis:SubordinateValueType"
minOccurs="0" >
            <xs:sequence>
            <xs:attribute name="Value" use="required" type="xs:ID" / >
 </xs:complexType>
<!-- -->
<xs:complexType name="SubordinateValueType">
            <xs:attribute name="Value" use="required" type="xs:IDREF"/
</xs:complexType>
<!-- -->
<xs:complexType name="DomainsType">
            <xs:sequence>
        <xs:element name="Domain" maxOccurs="unbounded" type="permis:DomainType" />
            </xs:sequence>
</xs:complexType>
<!-- -->
  <xs:complexType name="DomainType">
            <xs:sequence>
            <xs:element name="RootNode" type="permis:RootNodeType"
maxOccurs="unbounded"
            <xs:sequence>
            <xs:attribute name="DomainID" use="required" type="xs:ID"/ </xs:complexType>
<!-- -->
 <xs:complexType name="RootNodeType">
            <xs:sequence>
            <!-- the excluded nodes must be immediately subordinate to the root node.
            Only LDAP and HTTP URLs are currently allowed for nodes -->
            <xs:element name="ExcludedNode" type=" xs:anyURI " minOccurs="0"
maxOccurs="unbounded"
            <xs:sequence>
            <xs:attribute name="Name" type="xs:anyURI" use="required"/>
</xs:complexType>
<!-- -->
<xs:complexType name="AttributeAssignmentsType">
            <xs:sequence>
            <xs:element name="AttributeAssignment" maxOccurs="unbounded"
type="permis:AttributeAssignmentType"/>
            </xs:sequence>
</xs:complexType>
<!-- -->
<xs:complexType name="AttributeAssignmentType" >
             <xs:sequence>
            <xs:element name="Attribute" type="permis:AttributeType" minOccurs="0"
maxOccurs="unbounded" />
             </xs:sequence>
            <xs:attribute name="AAID" use="required" type="xs:ID"/>
            <xs:attribute name="TI" use="required" type="xs:IDREF"/
            <xs:attribute name="DomainID" use="required" type="xs:IDREF"/>
            <xs:attribute name="DelegationDepth" use="optional"
type="xs:nonNegativeInteger"/>
   </xs:complexType>
<!-- -->
```

```
<xs:complexType name="AttributeType">
        <xs:sequence>
        <xs:element name="AttributeValue" type="permis:SubordinateValueType"
minOccurs="0" >
        <xs:sequence>
        <xs:attribute name="FriendlyName" use="optional" type="xs:IDREF"/
</xs:complexType>
<!-- -->
</xs:schema>
```