“SPADE: **SP**KI/SDSI for **A**ttribute Release Policies in a **D**istributed **E**nvironment”

Sidharth Nazareth

DARTMOUTH COLLEGE

Hanover, New Hampshire

May 30th, 2003

Dartmouth Computer Science Department Technical Report TR2003-453

Abstract

Shibboleth is a federated administrated system that supports inter-institutional authentication and authorization for sharing of resources. SPKI/SDSI is a public key infrastructure whose creation was motivated by the perception that X.509 is too complex and flawed. This thesis addresses the problem of how users that are part of a Public Key Infrastructure in a distributed computing system can effectively specify, create, and disseminate their Attribute Release Policies for Shibboleth using SPKI/SDSI. This thesis explores existing privacy mechanims, as well as distributed trust management and policy based systems. My work describes the prototype for a Trust Management Framework called SPADE (SPKI/SDSI for Attribute Release Policies in a Distributed Environment) that I have designed, developed and implemented. The principal result of this research has been the demonstration that SPKI/SDSI is a viable approach for trust management and privacy policy specification, especially for minimalistic policies in a distributed environment.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

According to [Clarke],

“The Internet has facilitated growth in the development of distributed computing systems”.

In order to handle the large number of users in these systems in an efficient manner, scalable security schemes such as a Public Key Infrastructure (or PKI) have been developed. Today, the X.509 Public-Key Infrastructure has emerged as the de facto standard for PKIs. However, recently SPKI/SDSI [Ellison2] is another PKI standard that has been proposed. SPKI/SDSI is motivated by the perception that X.509 is too complex and lacking on a number of issues, such as its lack of support for authorization.

As users access more and more resources on the Internet, online privacy has naturally become a major concern. According to [wwwP3P],

"Consumer polls have consistently demonstrated that privacy protection is a significant concern and are expressing concern about what data is collected from them."

Individuals want more control over how their personal information is gathered and used. This type of privacy is called *Data Privacy* or *Information Privacy* which according to Jeffey Sheldon in[Sheldon] is,

“the concern about an individual’s control over personal information about him or her.”

However, users are often willing to disclose personal information online in order to obtain extra services. At the same time, the temptation to release information may be damaging to the organization or the institution that the user and his attributes are a part of. Thus, not only is there a need for a mechanism by which a user can specify his privacy preferences or policy, but also for a mechanism by which an organization or institution can specify its own privacy preferences or policies which are applicable to its different users acting in varied roles, and which can serve as a filter or overwrite a user's potentially damaging privacy policy.

Shibboleth[ShibDraft] is a federated administrated system (where authentication and authorization is handled by both the origin site and the resource site) that supports inter-institutional authentication and authorization for sharing of resources, available to users from those institutions. Shibboleth assumes that users employ standard web browsers to access these resources. Shibboleth securely transfers attributes about a user from the user's origin site to a resource provider site. Shibboleth places a large emphasis on user privacy and even allows its users a choice in what information gets released about them and to which site. This, therefore, places the responsibility of balancing access and privacy in the hands of the user.

This thesis addresses the problem of how users that are part of a Public Key Infrastructure in a distributed computing system can effectively specify, create, disseminate and change their respective Attribute Release Policies for Shibboleth using SPKI/SDSI. This thesis explores existing privacy mechanims, as well as distributed trust management and policy based systems. I describe the prototype for a Trust Management Framework called SPADE (SPKI/SDSI for Attribute Release Policies in a Distributed Environment) that I have designed, developed and implemented. The principal result of this research has been the demonstration that SPKI/SDSI is a viable approach for trust management and privacy policy specification, especially for minimalistic policies.

1.2 Problem Statement

My solution addresses the following problem:

"How might an individual user of an organization's distributed system (such as its public key infrastructure) specify, create and disseminate an Attribute Release Policy (ARP) that can be used by the the Shibboleth system, and how does that ARP interact with the host organization's own ARP?"

1.3 My contribution

My work includes the design, development and implementation of a prototype of SPADE (SPKI / SDSI for Attribute Release Policies in a Distributed Environment). The SPADE infrastructure consists of logically dividing an institution or organization into domains that may also correspond to physical or structural divisions, such as project groups or academic departments. Each such domain is defined by an administrator who is in charge of distributing and managing trust for that domain, and a group of users who belong to that domain. Users within the organization may be members of more that one domain, but administrators may only be affiliated with one domain. Trust is established and managed using SPKI/SDSI public key certificates.

I have connected SPADE to a Shibboleth test deployment at Dartmouth for my initial demonstration. SPADE acts as an extension of the Shibboleth Attribute Authority (AA). When standard requests for user attributes come into the AA, they are redirected to SPADE which contacts the relevant user domain and obtains the user's ARP. This ARP is combined with other ARPs in the organization by obtaining a chain of certificates (where each certificate is an attribute release policy) from the relevant domains of the organization starting with the individual domain user and his administrator. These ARPs are intersected to derive the final ARP. By extending the AA function in this way, this work does not involve modifying any part of the Shibboleth standard.

I have modified and used SPKI/SDSI code developed at MIT [MIT] for a main part of my work. Important additions I have made include writing the tag intersection library which follows the SPKI/SDSI standard [Ellison2], which was not written previously.

I have also implemented a web-based GUI that allows the domain users and administrator to log on and manage their policies. For the administrator, this involves authorizing new users in the system, creating default attribute release policies, and creating roles for the different users to be assigned to. A user may use the GUI to create and modify his attribute release policy.

The principal goal of SPADE is to show the viabililty and effectiveness of using SPKI / SDSI as a basis for a distributed trust system for specifying and conveying policies.

1.4 Thesis Organization

This thesis is organized as follows:

- Chapter 2 discusses the background to this work. This includes certificate standards such as X.509 and SPKI/SDSI. I also discuss privacy, policy concepts and the Shibboleth system
- Chapter 3 describes the characteristics, design and implementation of SPADE.
- Chapter 4 describes related work done in this area. These include privacy mechanisms, policy languages, and trust management systems.
- Chapter 5 provides a summary of my work
- Chapter 6 concludes the thesis with a discussion of opportunities for future work in this area.

2 Background

2.1 Privacy concepts

Security technologies and the protection of individual's privacy are closely intertwined. While an organization's privacy goals may be expressed using human-readable, machine-enforcable policies (such as Shibboleth Attribute Release Policies), security technologies are typically responsible for enforcing those policies. But security technologies may also be applied more broadly, to problems of authentication and authorizaton. Some of the common definitions of privacy related terms are enumerated by Paul Madsen and Carlisle Adams in [Madsen1], and are given below:

> *"**Confidentiality** refers to keeping sensitive information secret and protected from inappropriate viewing. Privacy requires that the confidentiality of user information is protected both in transit and in storage."*
>
> *"**Authorization** refers to the process of determining what an individual is allowed to do."*
>
> *"**Authentication** refers to proving that individuals are indeed who they claim to be."*
>
> *"**Privacy** of user information requires: 1)Protected data storage, 2)Authentication and authorization of requesting applications, and 3) Confidentiality of transmitted data."*

2.2 Policy concepts

According to N. Dulay et al in [Dulay],

> *" A Policy is a set of rules that govern the choices in the behavior of a system"*

Typically, policies in system are of two types, as mentioned by J. Vollbrecht et al in [Vollbrecht]:

1. **Security policies**: These specify what the response to a query should be in a system.

2. **Management policies**: These oversee the system and specify how a query must be handled in order for a security policy to be applied to it.

There exists a strong relationship between authorization and policy. Authorization is usually granted or denied based on a policy. In order for policies to be effective in decided authorization, they must go through three steps, as mentioned by J.Vollbrecht et al in [Vollbrecht]:

1. They must be **retrieved**. Policies may be dispersed through the organization. They are retrieved using a *Policy Retrieval Point* (or *PRP*).

2. They must be **evaluated**. Rules commonly exists for combining policies and sets of policies. These are used to evaluate the policy and reach a decision. These are commonly done at a *Policy Decision Point* (or *PDP*).

3. They must be **enforced.** This usally entails granting or denying an authorization. This is done at a *Policy Enforcement Point* (or *PEP*).

2.3 Public Key Infrastructures

According to Charlie Kaufman et al in [Kaufman],

> *"A Public Key Infrastructure (or PKI) is a collection of mechanisms for creating, distributing and using public keys"*

The main purpose of a public key infrastructure, as the name suggests, is to specify how names and attributes should be bound to public keys. This is done by using a public key certificate which is a signed statement binding 'something' to a public key. Typically, in any PKI, there are two types of certificates - an *Identity Certificate* and an *Authorization certificate*. An Identity Certificate usually binds a public key to a name. An Authorization

Certificate binds a public key to attributes and signifies an authorization or privilege that is meaningful to the signer and some set of interpreters.

Of the existing Public Key Infrastructures in use today, X.509 certificates are the most widely used, and have emerged as the *de facto* standard. X.509 uses designated *Certification Authorities* (or *CA*) that have the authority to sign public-key certificates for entities in the system. These CAs thus act as 'trust roots' and verify that an entity (usually a user) is the rightful holder of a particular public key.

More recent public schemes proposed, such as SPKI/SDSI move away from the traditional idea of using an individual's (allegedly) unique global name and binding a public key to it.

2.3.1 X.509

According to [Clarke], *"X.509 is the conventional Public Key Infrastructure"*.

An X.509 principal is a user's (allegedly) globally unique public key. This name is called a *Distinguished Name* (or *DN*). An X.509 certificate thus binds a user's DN to a public key to produce an Identity certificate. X.509 is based on a hierarchical global namespace. There are designated Certification Authorities (or CAs) that sign and issue certificates to X.509 users. Thus, as quoted from [Clarke], "X.509 communities are built from the top-down, with trust extending from root CA keys." Trust is established by examining a chain of certificates from a trusted CA's key to a user's public key.

Recently, *Version 3 of X.509* (or *X.509v3*) has been released which contains support for specifying attributes and authorization. This was done by providing a means of specifying additional fields in an X.509 certificate where attributes could be specified. Thus, an X.509v3 certificate may contain a binding between a Distinguished Name and a public key and a Distinguished Name and a set of attributes.

However, there are a number of problems with X.509 in general, especially with regard to authorization. Some of them are as follows:

1. By binding a user's name to a public key, and then binding that same name to a set of attributes all in one X.509v3 Certificate, we immediately compromise the user's anonymity. Moreover, while user's names are usually bound to their public keys for long periods of time, attributes are usually bound to users for short periods. Thus, if a user were to want to change his attributes, it would mean revoking his 'identity' too!

2. To solve the above problem, attributes are specified in a seperate certificate called an X.509 *Attribute Certificate* (or *AC*). Typically, this certificate binds a subject's Distinguished Name with a set of attributes and is presented to the verifier of the subject's attributes together with the subject's X.509 Identity Certificate. This incurs the overhead of creating two X.509 certificiates each time for conveying a subject's attributes. Moreover, since the principal of both the Identity Certificate and the Attribute Certificate is a name, the entity that verifies the authenticity of the user may be the same entity that verifies his authorization. This is usually not desired. A more serious security violation would be just the X.509 Attribute Certificate being inspected, without consulting the subject's Identity Certificate.

3. X.509 data formats are expressed in the *Abstract Syntax Notation One* (or *ASN.1*) standard. Most people cite is as human unreadable. Another problem is it's complexity, arising from the fact that it is very powerful. Thus, X.509 is complex for application and service developers to implement.

4. All CAs are not equal. Besides the risk of a CAs key being compromised and therefore comprimising its entire 'domain', different commercial CAs offer different levels of service to their clients as specified in their *Certificate Practice Statement* (or *CPS*). The CAs CPS is basically the procedures and rules that the CA follows. While some consider it good that a CAs CPS is not part of the standard and that different CAs can

be flexible in specifying their own rules, the lack of a common mechanism results in different CAs providing a different level of service and consequently a different trust level. Moreover, a CAs CPS is specified in convoluted language with legal jargon that hardly anyone but the CA cannot understand.

For all these reasons, a number of new PKI standards were proposed, especially for authorization and public key centricity.

2.3.2 PGP

Pretty Good Privacy (or *PGP*) was released by Phil Zimmerman in the early 1990s. It has the following characteristics, as described by Dwaine Clarke in [Clarke]:

1. It is **egalitarian**: There is no hierarchical structure in PGP and all public keys are free to issue certificates. In comparison to X.509, every public key in PGP is a Certification Authority.

2. Binding **names** to public keys: PGP also suffers from the flaws of X.509 where it looks to bind a global name to a public key. This is usually a user's email address.

3. *"Web of Trust"*: PGP users build paths of trust among themselves in a distributed manner. Users may trust other users to vouch for the authencity of certain public keys. Thus, PGP users actually function as CAs themselves, while still allowing any user to issue certificates for any other PGP user. Consequently, there are a number of certification paths leading through different CAs. Trust is truly distributed in this case.

While PGP may be fine for cases where a lot is not at stake such as e-mail, it is not feasible for systems which handle complex trust relationships where a large number of users are involved and who cannot be trusted to make the right choices.

2.3.3 SPKI/SDSI

The Simple Distributed Security Infrastructure (or *SDSI*) [Rivest1] was designed in 1996 by Ron Rivest and Butler Lampson at the Massachusetts Insititute of Technology. Its main goal, according to [Clarke], was *"to facilitate the building of secure, scalable, distributed computing systems"*. Around the same time, the *Simple Public Key infrastructure* (or *SPKI*) [Ellison2] was proposed by Carl Ellison for a simple public key infrastrucute authorization model. These two proposals were merged in 1998 to form SPKI/SDSI.

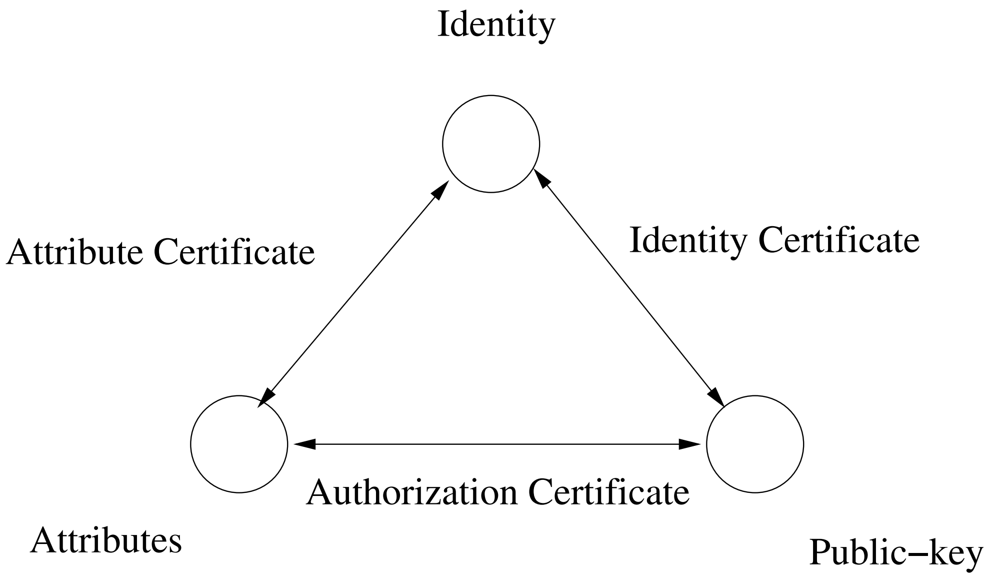


Figure 1: Carl Ellison's Authorization triangle

One of the main reasons SPKI/SDSI was proposed was to provide a flexible and lightweight authorization model as compared to X.509. Moreover, the model was to be public key centric so that principals would be truly unique. As seen from Figure 1 (which is often called Carl Ellison's *Authorization triangle* and is proposed by him), SPKI/SDSI works by combining a public key with a set of attributes to form an authorization certificate, unlike its counterpart X.509 which forms an Attribute Certificate by binding a Distinguished Name to a public key. SPKI/SDSI can also create Identity certificates binding names to public key but they are not used for authorization decisions.

2.3.3.1 Main characteristics of SPKI/SDSI

1. **Key-centric:** SPKI/SDSI principals are public keys. All authorization binding and verification is done with public keys and attributes only, not with identities such as names.

2. **Egalitarian:** There is no global hierarchy or hierarchical infrastructure necessary in SPKI/SDSI. Every principal is free to issue certificates to other individuals. This also leads to scalability.

3. **Certificate types:** There are two types of certificates in SPKI/SDSI (like X.509). These are:

   (a) Name Certificate: This binds a local name and a public key. The name is valid only for the local namespace and is used by the principal only for personal identification. It is never used globally. There are four fields to a SPKI/SDSI name certificate. These are the issuer, an identifier such as a local name, a subject and validity dates.

   (b) Authorization Certificate: This binds a public key to attributes. It is used in authorization decisions. There are five fields in an authorization certificate. These are the issuer, the subject, the propogate flag (or not) which specifies whether this subject of this certificate can delegate (or not), and validity dates.

4. **Groups:** SPKI / SDSI has support for creating groups and is one of its main advantages. This is done by issuing multiple (name , key) certificates with the same name - one for each group member. The concept of groups can also be used to create roles for role based authorization.

5. **Delegation:** One of the other main advantages of SPKI/SDSI is the ability to delegate authorization. This is specified using the propogate bit in the Authorization Certificate. If this bit is set, then the subject of this certificate can re-delgate the permission or attributes he is given. Delegation is transitive in that if Alice delegates to Bob (with permission to delegate further) and Bob delegates to Carol, then in effect Alice has delegated to Carol.

6. **Authorization flow:** An authorization flow or chain is a chain of valid certificates that specifies an authorization. According to [Clarke], *"an authorization flow can consist of just authorization certificates or both name certificates and authorization certificates"*. The 'trust root' of an authorization flow is the first principal in the chain who started the delegation process. This is in marked contrast to the X.509 model with CAs as trusted roots.

7. **S-expressions:** SPKI/SDSI certificates uses S-expressions which are *"human-readable ASCII representations" [Rivest2].* This makes it easy for humans to read a certificate and deduce it's meaning. This is in contrast to X.509 which uses ASN.1.

2.3.3.2 Advantages of SPKI/SDSI over X.509[1]

1. As mentioned in [Ellison2], *"The main purpose of SPKI/SDSI certificates is authorization, as opposed to the purpose of X.509 certificates which is authentication."*

2. SPKI/SDSI certificates use S-expressions as the standard format for authorization attributes, whereas X.509 uses the complex ASN.1 standard. SPKI/SDSI also defines a canonical form for S-expressions and a mechanism for deriving authorization decisions. According to noted computer security expert Carl Ellison [Ellison3], parsing canonical S-expressions take much less code size that parsing ASN.1 data. (According to Carl, his S-expression parsing code takes 8 KB. On the other hand, his compiler

[1]Some of the text in this paragraph 2.3.3.2 is quoted verbatim from my Master's Thesis Proposal [Nazareth].

to parse ASN.1 takes 50 KB and some commerical ASN.1 compilers take 500 KB of code). Thus, S-expressions are arguably faster to parse and compile than ASN.1.

3. SPKI/SDSI uses a 'tag-language' that is used to express authorization information through large sets of strings or S-expressions. Standard libraries may be written to intersect certificate tags and deduce what an authorization chain authorizes. On the other hand, because of X.509's generality each developer is free to define an X.509v3 extension format and then define how those attributes may be intersected. Thus, each application using X.509 has to write a seperate X.509 chain-processing routine.

4. Groups or Roles can be defined in SPKI/SDSI by, as mentioned in the previous subsection, issuing multiple (name , key) certificates with the same name - one for each group member.

5. SPKI/SDSI can implement threshold certificates where authorization is a collective decision of k-of-n entities.

6. SPKI/SDSI makes it easy to issue short lived certificates as it does not have *Certificate Revocation Lists* (or *CRLs*).

7. To quote from [Nykanen],

   "When the usage of PKI is authorization, then SPKI is conceptually a better choice, because not having to care about names at all simplifies the process of certification and validation remarkbly...When it comes to plain anonymous authorization, especially when the entities are not controlled by humans....., SPKI certificates would be the better choice. The possibility to allow several paths for certificate validation, and having a control on the acceptable level of successively validates paths make SPKI favorable..the strict hierarchy of PKIX (i.e using X.509) is very sensitive. SPKI has been designed in such a way that interoperability with other PKIs has been taken into account."

8. In one of my favorite quotes from noted computer security expert Carl Ellison, he says,

   "One advantage SPKI has is that our names are global because they're local, while X.509 names are local because they're global..That is, because we use SDSI local names but need global names, we made the local names global by pairing local names with keys (or key hashes) and we got truly global names. X.509, by contrast, was designed to use globally unique X.500 Distinguished Names (DN). Therefore, they use those names alone. However, X.500's global name space was a delusion that will never come to pass, so what happens instead is that each CA generates its own Distinguished Names. Those DNs are therefore local because they came from the mind of the CA and not some global authority (X.500). Because they're not paired with keys the way SPKI's SDSI names are, the names used in X.509 attribute certificates are truly local (and therefore not unique and therefore a security flaw)".

2.3.4 Comparison of X.509, PGP and SPKI/SDSI

A comparison of the three public key infrastructures are shown in Table 1 (as taken from [Clarke]).

X.509	Name Space:	Global
	Types of Certificates:	Name Certificates
	Name-to-Key binding:	Single-valued function: each global name is bound to exactly one key (assuming each user has a single public-private key pair).
	CA Characteristics:	Global Hierarchy. There are commercial X.509 CAs. X.509 communities are built from the top-down.
	Trust Model:	Hierarchical Trust Model. Trust originates from a trusted CA, over which the guardian may or may not have control. A requestor provides a chain of authentication from the trusted CA to the requestor s key.
	Signatures:	Each certificate has one signature, belonging to the issuer of the certificate.
	Certificate Revocation:	Uses CRLs
PGP	Name Space:	Global
	Types of Certificates:	Name Certificates
	Name-to-Key binding:	Single-valued function: each global name is bound to exactly one key (assuming each user has a single public-private key pair).
	CA Characteristics:	CA Characteristics: Egalitarian design. Each key can issue certificates. PGP communities are built from the bottom-up in a distributed manner.
	Trust Model:	Web of Trust
	Signatures:	Each certificate can have multiple signatures; the first signature belongs to the issuer of the certificate.
	Certificate Revocation:	A suicide note is posted on PGP certificate servers, and widely distributed to people who have the compromised key on their public keyrings.
SPKI/SDI	Name Space:	Local
	Types of Certificates:	Name Certificates, Authorization Certificates
	Name-to-Key binding:	Multi-valued function: each local name is bound to zero, one or more keys (assuming each user has a single public-private key pair).
	CA Characteristics:	Egalitarian design. The principals are the public keys. Each key can issue certificates. SPKI/SDSI communities are built from the bottom-up in a distributed manner.
	Trust Model:	Trust originates from the guardian. A requestor provides a chain of authorization from the guardian to the requestor s key. The infrastructure has a clean, scalable model for defining groups and delegating authority.
	Signatures:	Each certificate has one signature, belonging to the issuer of the certificate.
	Certificate Revocation:	Advocates using short validity periods and Certificates of Health.

Table 1: Comparison of X.509, PGP and SPKI/SDSI [Clarke]

2.4 Shibboleth

From the Shibboleth draft [ShibDraft],

> *"Shibboleth is an Internet2/MACE project with intellectual and financial support from IBM which involves developing architectures, frameworks, and practical technologies to support interinstitutional authentication and authorization for sharing of resources"*

One of Shibboleth's main aims is to accomodate the different security systems existing in organizations and college campuses today. It is a middleware architecture that allows users from an 'origin' site to access resources at a 'target' site.

The two main efforts of the Shibboleth project are:

1. To develop an architecture so that users in different organizations may be able to authenticate across boundaries and share resources, and
2. To promoting interoperability by using standards, such as SAML and documented methods of information exchange.

Shibboleth places a great importance on user privacy. It enables users to be able to access resources anonymously by presenting only their attribute values and an origin site issued *'handle'* to the target site.

2.4.1 Shibboleth components

Figure 2 (taken from the Shibboleth draft) shows the main Shibboleth components. The following are the main components (described in more detail in the Shibboleth draft [ShibDraft]) that allow users to access resources across institution or organization boundaries:

SHIRE (or Shibboleth Indexical Reference Establisher): The SHIRE is the first component that the user contacts (transparently) when he tries to access a resource on the target

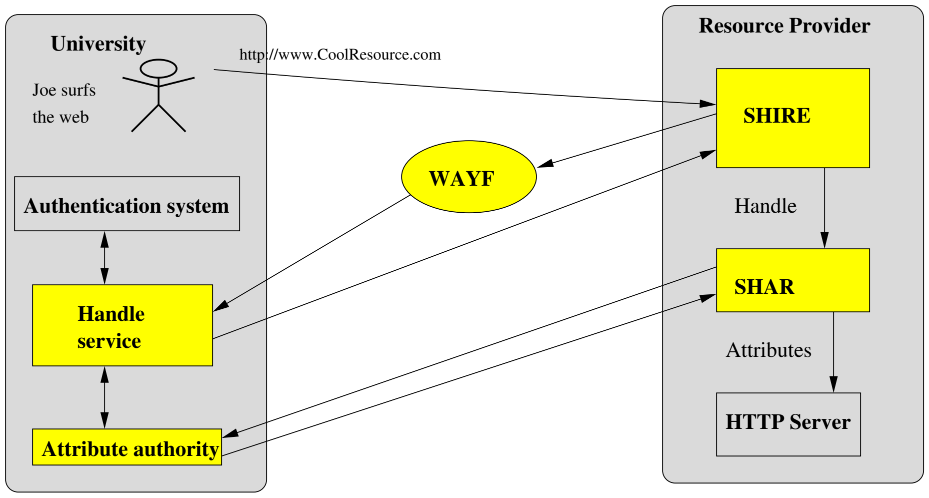


Figure 2: Shibboleth components [ShibDraft]

site. The SHIRE is usually a web server that waits for users to connect to it through their browsers and obtains a handle for a user that target side uses later to be able to obtain user attributes.

WAYF (or Where Are You From?): The SHIRE contacts the WAYF component to obtain a handle for the user from the user's *'Handle Service'* (or *HS*). The WAYF component actually interacts with the user and asks him where he is from (The user usually has to select from a drop down menu. In this way, the WAYF can also control which institutions have access to the target site). The WAYF stores the mapping between the user's origin site and the URL of the user's HS, and is usally a part of the target site.

HS (or Handle Service): The Handle Service is an origin side component. It is responsible for making sure that the user has authenticated locally. It also creates an anonymous handle that the target site later presents to the origin site to retrieve attributes about the user. The HS returns the handle to the WAYF along with the URL of the user's Attribute Authority (or AA).

SHAR (or Shibboleth Attribute Requester): The SHAR is a target site component that is responsible for retrieving the user's attributes to enable resource access. After the SHIRE passes the user's handle to the SHAR, the SHAR presents the handle to the user's Attribute Authority to obtain his attributes.

AA (or Attribute Authority): This is a major component of Shibboleth. It sits on the origin site and handles requests for attributes from the resource provider's site. The AA is not part of the Shibboleth standard. There may be multiple AAs at an origin site but must be at least one. The AA must provide users at the origin side a means by which they can specify their Attribute Release Policies (or ARPs). This is usally done using a GUI such as a web browser and enables the user to control his own privacy. The downside of course, is that a user's choice of ARP may not be proper to be able to grant him access to a target resource. Due to this, it is often preferable for the user to be aware of each sites attribute requirements, possibly shown on the interface. The process of finding the user's attributes and sending them to the target site is usually done in two steps:

1. The handle presented to the AA from the SHAR is mapped to the user's identity. This is usually done using a handle cache which stores all recently created handles.

2. The user's identity is then used to retrieve his ARP. This process is not part of the Shibboleth standard and is left open to interpretation. The user may have a single ARP or multiple ARPs. They may be dispersed throughout the organization or they may be collected in one place. How the user's ARP is retrieved, validated and enforced is left to the implementers. My model, SPADE, chooses a distributed ARP setting using SPKI/SDSI certificates to store ARPs.

RM (or Resource Manager): After the SHAR receives the user's attributes from the AA, it sends them to the manager of the resource that the user is trying to access. The RM may have his own Attribute Acceptance Policy (or AAP) which decides whether the user wil be granted or denied access to the resource based on the attributes presented.

2.4.2 Shibboleth Attribute Release Policies

Shibboleth attributes are usually name/value pairs. Examples include “Role=Physics Student”, “Name=Jane Smith” and “Email=jane.smith@college.edu”. An important point to note is that Shibboleth target sites are greedy and will try to obtain as many of the user’s attributes as possible. An ARP is used by a AA to retrieve attributes about a particular user. Usually, the ARP used is one which the user created himself. Shibboleth ARPs consist of three main fields:

1. A destination SHAR name
2. A URL
3. A list of attributes that the user specifies to be released to this SHAR and URL

The SHAR is usually the website where the URL resides and which hosts the resource. An example of a SHAR-URL pair is

“SHAR=http://www.mit.edu, URL=http://www.mit.edu/ai/reconfiguring.jsp”

An important point to note is that a user must only be allowed to release attributes which are relevant to him, and they must be released using ARPs. A student thus cannot release faculty attributes and so on. This constraint can be imposed in a number of ways. The AA may serve to check attributes leaving it against the user’s identity. Else, other entities in the organization, such as the database that stores the user’s ARP, may serve to check the attributes being released. Thus, not only can a user have his own ARP, but other local authorities in the organization can serve to impose their own ARPs on the user. There is also the possibilty of institutional ARPs and hidden attributes being released. For example, Dartmouth College may have a contractual agreement with the Smithsonian institute in

Washington D.C that allows Darmouth users to access it. However, Dartmouth must provide additional information that the users have no knowledge of and concern about, such as a contract number or a user statistic. Thus, the AA may serve to add additional attribute values into the Attribute Response Message (or ARM) sent to the SHAR. The Shibboleth draft [ShibDraft] also states that AAs must provide their users with a web based GUI so that they can easily specify and modify their ARPs.

Default and Wildcarded ARPs: Since is not possible for a user and his institution / organization to be able to provide ARPs for every target resource on the Internet, and every SHAR and URL pairing, there must exist a mechanism by which users can specify default ARPs. This saves time and can avoid denial of service to the user if the user chooses an intelligent default ARP. Additionally, a user can create a wildcarded ARP that gives him more control over which sites he chooses to release certain attributes to. For example, one a typical wildcarded ARP is shown in Figure 3. This ARP specifies that the values of the user's two attributes of Role and Name can be release to any site that the user accesses at the University of Wisconsin.

SHAR: http://www.wisc.edu

URL: http://www.wisc.edu/*

ATTRIBUTES: Role='Student', Name='John Doe'

Figure 3: Example of Shibboleth Wildcarded ARP

2.4.3 Shibboleth event and message flows

Figure 4 [ShibDraft] shows the sequence of events and message flows when a user accesses or tries to access a Shibboleth managed resource.

The steps are as follows:

1. A user at an origin site uses a web browser to access a Shibboleth managed resource

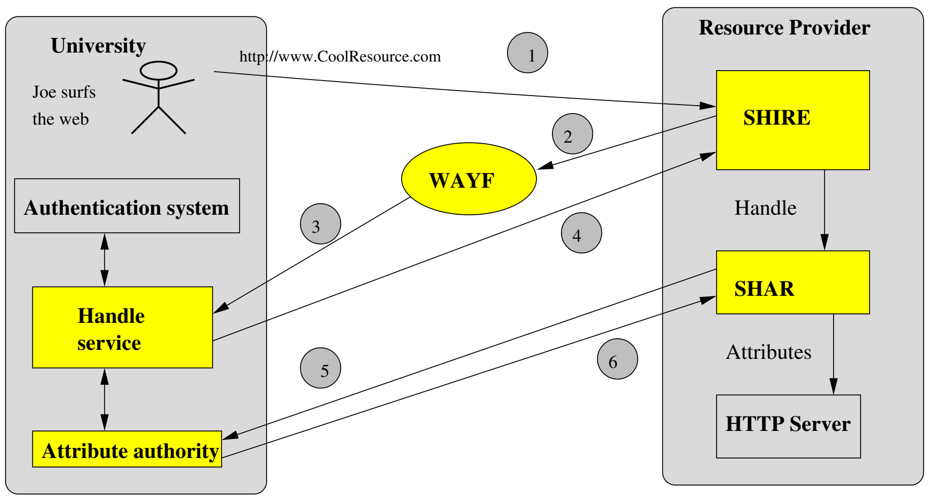


Figure 4: Shibboleth event and message flows [ShibDraft]

residing on a HTTP server.

2. The SHIRE redirects the user's request to the WAYF.

3. The WAYF contacts the user's HS. The user authenticates at the origin site, an anonymous handle is created and the HS receives it.

4. The handle is sent to the SHIRE along with URL of the user's AA. The SHIRE passes the handle to the SHAR.

5. The SHAR contacts the user's AA and asks for user attributes using an *AQM* (or *Attribute Query Message*).

6. The AA consults the user's ARP, and uses it to retrieve the relevant user atttributes. It then passes these to the SHAR using an *ARM* (or *Attribute Response Message*).

It is important to note, as stated in the Shibboleth draft [ShibDraft],

"AA implementers are free to support many different kinds of ARPs with varying semantics as long as the AA can efficiently process requests and determine

the effective policy to apply...Shibboleth doesn't specify or constrain how an AA can answer these kinds of questions."

Relation with SAML (or Security Assertion Markup Language): Shibboleth uses SAML for it's AQM and ARM formats. A description of SAML is given in 4.2.3.

3 My Solution: SPADE

As stated in section 1.2 I address the following problem,

"How might an individual user of an organization's distributed system (such as its public key infrastructure) specify, create and disseminate an Attribute Release Policy (ARP) that can be used by the the Shibboleth system, and how does that ARP interact with the host organization's own ARP?"

My solution, SPADE, explores how SPKI/SDSI may be used to solve this problem. The main concept behind SPADE is that different users in an organization use signed SPKI/SDSI Authorization certificates to specify their ARPs. When a Shibboleth Attribute Query Message (or AQM) comes in, SPADE retrieves and intersects a number of these SPKI/SDSI ceritificates to derive a resultant ARP. This ARP is used as a basis for retrieving the user's attributes from a database, which are then sent to the target site.

I now describe the characteristics, design and implementation of SPADE.

3.1 SPADE characteristics

After a careful review of related work, enumerated in Chapter 4 and summarized in Table 3, I have selected the best features to incorporate into SPADE. These are given below:

3.1.1 Authorization framework

SPADE uses the SPKI/SDSI standard [Ellison2], which is designed for lightweight distributed delegation and authorization. SPADE's concern is authorization, not authentication. The user is assumed to be already authenticated when the system uses SPADE to retrieve the user's relevant ARPs. This is in keeping with the Shibboleth model. SPADE is public-key centric and attributes are bound to a public key resulting in an authorization certificate.

3.1.2 Trust Management

According to Kagal et al [Kagal],

> "SPKI was the first proposed standard for Distributed Trust Management"

SPADE, which uses SPKI/SDSI, possesses many of the properties of a Distributed Trust Management system such as delegation and authorization.

SPADE can also be thought of as a SPKI/SDSI *Privilege Management Infrastructure* (or *PMI*) [Chadwick2] that uses SPKI/SDSI authorization certificates instead of X.509 attribute certificates. Like PERMIS, SPADE embeds the policy in the certificate. However, unlike PERMIS, I use S-expressions and the standard SPKI/SDSI tags defined in [Ellison2] instead of an XML based policy. More information on PMI and PERMIS can be found in 4.5.

3.1.3 Role Based Authorization

SPADE uses SPKI/SDSI Name certificates to create roles for groups of users. It is similar to a Trust Establishment Framework [Herzberg].

As stated by the developer's of IBM's Trust Establishment Framework,

> *"A Trust Establishment system takes as input a user Attribute Certificate, and identifies a role based on a policy mapping from certificates to roles".*

In SPADE, subjects are identified by roles bound to their public keys. Each user is part of a domain. The administrator of this domain creates a SPKI/SDSI Name certificate to map a user to a role. A user creates his ARP based on what role he possesses. At the same time, the organization's ARP that applies to that user also depends upon the user's assigned role. For example, consider three roles in an organization called Dartmouth College - "Faculty", "Secretary" and "Student". Dartmouth may decide that only two of these three roles - 'Faculty' and 'Secretary' are allowed to release their credit card number to Shibboleth resources.

Thus, when creating his ARP, a user is already constrained in what type of ARP he may create based on his role (i.e one that prevents him from wanting to release his credit card number). Additionally, there must also be an organizational ARP which specifies whether a user acting in the capacity of a 'Faculty' or 'Secretary' role can release his credit card number at any given time. This allows the organization to be able to vary it's policy (especially temporarily) when it deems necessary. Consider the following example:

There are two users "Alice" and "Bob" who have just joined the domain of "History department". Alice is assigned a 'Student' role and Bob is assigned a 'Faculty' role. Now, when Alice creates her ARP, she does not have the option of specifying whether she wants to release her credit card number or not as this option is not available to her being in a 'Student' role. She may choose to release other attributes and creates her ARP. Bob, however, in the role of 'Faculty' has that option and chooses to release his credit card number to a particular site, say Amazon. Two weeks later, Dartmouth college learns of credit card fraud taking place at Amazon which its users, including Bob, are not aware of. So, Dartmouth modifies its ARP to prevent any of its users' credit card numbers from going to the Amazon website. Thus, even though Bob has been allowed to specify that he would like to release his credit card number to Amazon, and he has specified that it should be released to Amazon, Dartmouth's ARP prevents the attribute from going through to the site. A week later, the fraud has been detected and taken care of and Amazon has assured its users that it is safe to send personal information to the site again. Dartmouth now modifies its ARP again to allow all credit card numbers going to Amazon to be released once again. In the meantime, whether Bob chooses to do so or not remains his personal choice.

3.1.4 Policy Expression

SPADE uses SPKI/SDSI S-expressions and tags [Ellison2] to express and derive policies. Policies are obtained by combining individual policies from a chain of certificates (an authorization flow) using a standard SPKI/SDSI tag intersection library which I have written. All policies are embedded with attribute names only and not their values. Attributes are stored in a database in a local domain on the Shibboleth origin side. Thus confidentiality is maintained even if certificates are inspected by other users. When a request for attributes comes in from the Shibboleth target site, the user's ARP is intersected with the organization's relevant ARP certificates to derive a resultant ARP. It is only when the resultant ARP is derived that the values of the attribute names enumerated in the certificate are retrieved from the user's database and returned to the Shibboleth Attribute Authority. Note that the attributes are sent back to the target from the Attribute Authority the usual Shibboleth way - by embedding in SAML.

3.2 SPADE Design

SPADE divides an organization into a number of logically seperable *Domains*. Each of these Domains consist of one *Administrator* and zero or more users. A Domain is uniquely identified by its Administrator's public key. The Administrator is in charge of the domain and is responsible for managing the users in his domain, creating ARPs, and creating roles and assigning them to users. Moreover, the Administrator is also responsible for establishing inter-domain trust relationships with other Domains.

The various components of SPADE (Figure 5) are detailed below.

3.2.1 Domain

A Domain is a logical entity in SPADE which represents a set of users with an administrator and is part of an organization or institution. Domains are unique because they are defined by their Administrator's public key. A Domain consists of the following three types of entities:

- ONE Administrator
- ZERO or more users

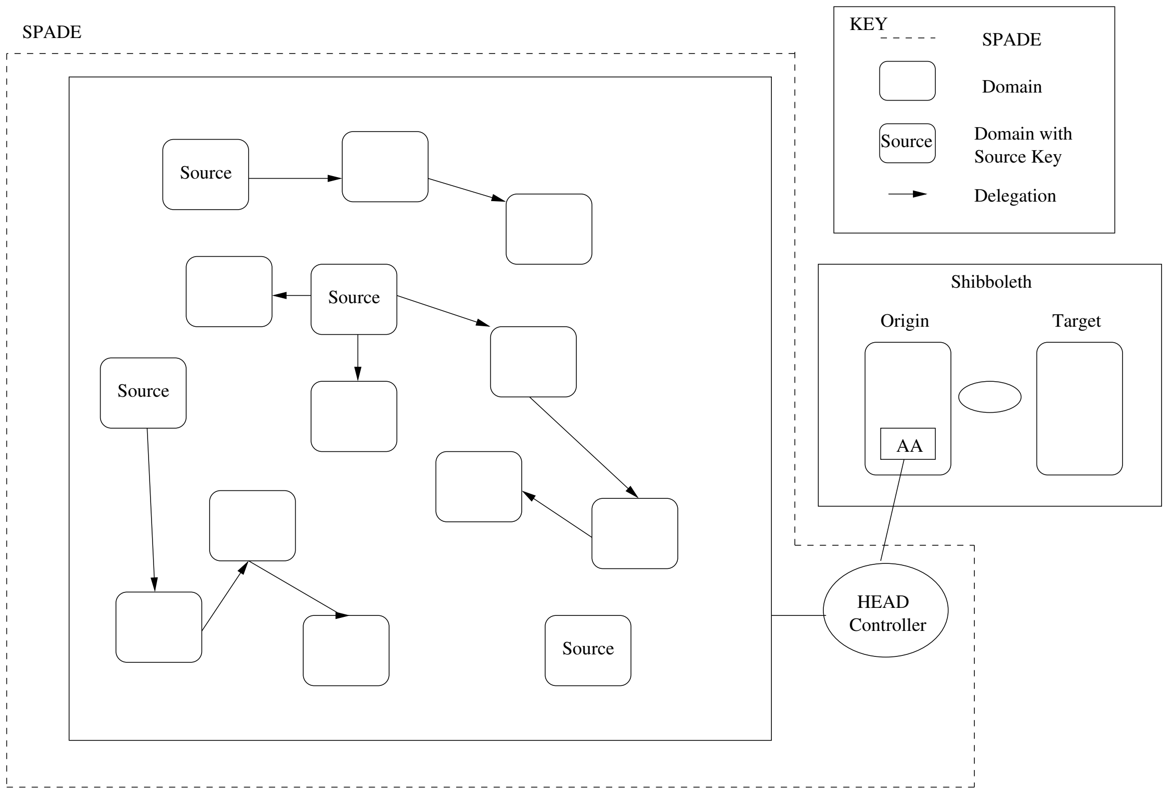


Figure 5: SPADE Architecture

- One *Domain Controller*

An examples of a Domain is the “History Department at Dartmouth”, the “IceCube Project in Distributed Systems at Microsoft Research”, or the “Computer Security Group at IBM”.

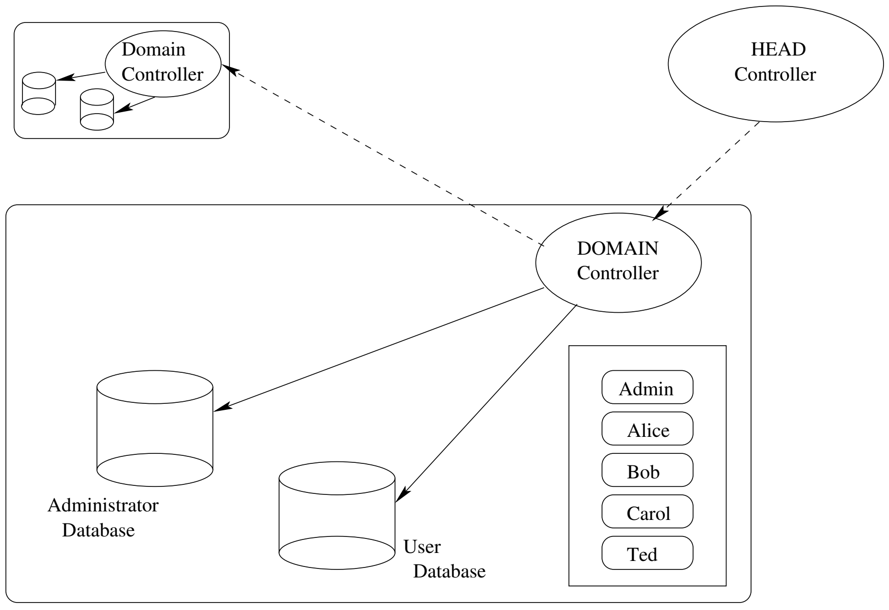


Figure 6: Head Controller with SPADE domains

The functions of the three domain entities are as follows:

Administrator:

There is a 1:1 relationship between a Domain and its Administrator. That is, each Domain has only one Administrator. This Administrator, in turn, can be affiliated to only this one Domain. The Administrator is responsible for the following functions:

1. Establishing Identity: The Administrator obtains the public key of the user (preferably offline) and then binds it to the User’s Name using a SPKI/SDSI Name certificate. This is solely for the purpose of identifying the owner of the public key when using the GUI. It is not used in the Authorization mechanism.

2. Creating Roles: The Administrator creates Domain roles using the SPKI/SDSI mechanism for creating groups. Thus, Authorization takes place based on the user's role, often called *Role Based Access Control* (*RBAC*).

3. Creating ARPs: The Administrator uses SPKI/SDSI Authorization Certificates to create ARPs. These include the Domain's own ARP (called a 'Filter' ARP) and an ARP to specify what type of ARP a user can create (called a 'Releasable' ARP).

4. Establishing inter-domain Trust Relationships: The Administrator establishes a trust relationship with the Administrators of two other Domains in the system, namely the '*Predecessor Domain*' and '*Successor Domain*'. Since Domains follow the hierarchical strucuture of the organization, the Predecessor Domain is the logical next higher Domain in the organizational hierarchy. The Successor Domain is the logical next lower Domain in the organizational hierarchy. For example, for the 'Arts and Science' Domain at Dartmouth, the Dartmouth Domain is it's Predecessor while the History Department is it's successor.

User:

There can be zero or more users in a domain. The users are responsible for the following:

1. Creating their Attribute Release Policies for different target resources in Shibboleth. Note that a Shibboleth target resource is specified by its SHAR and URL value pair. The SHAR is usually the website that the URL resides at. An example of a SHAR-URL pairing is

(SHAR:http://www.mit.edu, URL:http://www.mit.edu/robotics/reconfiguring.jsp)

Domains and trust relationships:

Apart from the trust established between an Administrator and the users in his Domain, there is also the trust between two Domains. The notion of trust in this case signifies that each Domain must believe that their Predecessor Domain and Successor Domain are who they say they are.Domains must trust each other because Attribute Release Policies from a number of them will be retrieved and intersected to form the final policy. These trust relationship starts at an organization 'Source' Domain and end with the user creating his ARP in his own Domain. A 'Source' Domain is the logical head of an organization. It usually does not contains users but is used to establish trust with other Domains through delegation. Trust, and consequently the organization's or institution's structure, is established in the following way:

- Each Domain defines a Predecessor Domain and a Successor Domain, in relation to its place in a certificate chain. As the name implies, a Predecessor Domain is one that delegates to this Domain, and a Successor Domain is one that this Domain delegates to. Each Domain Controller uses these relationships to retrieve ARPs and create a certificate chain. There are, of course, two exceptions to this. A Source Key Domain will not have a Predecessor. A Leaf Domain (where the user usually resides) will not have a Successor.

Domain types:

As shown in Figure 7, there are three types of Domains based on their position while forming a certificate chain. These are

1. Source Domain: This Domain forms the 'root' of the trust chain. It usually symbolizes the head of the organization. However this does not have to be the case. There may be multiple Source Domains. An example of a Source Domain is "Dartmouth Root", "IBM Research Head" or "Oracle CFO Office". For example, at Dartmouth, the Arts and Science College may be identified as a Source Domain. If there exists a delegation path starting with the 'Arts and Science' Domain Administrator's public key and ending with a user's, say Alice's, public key at the History department, then

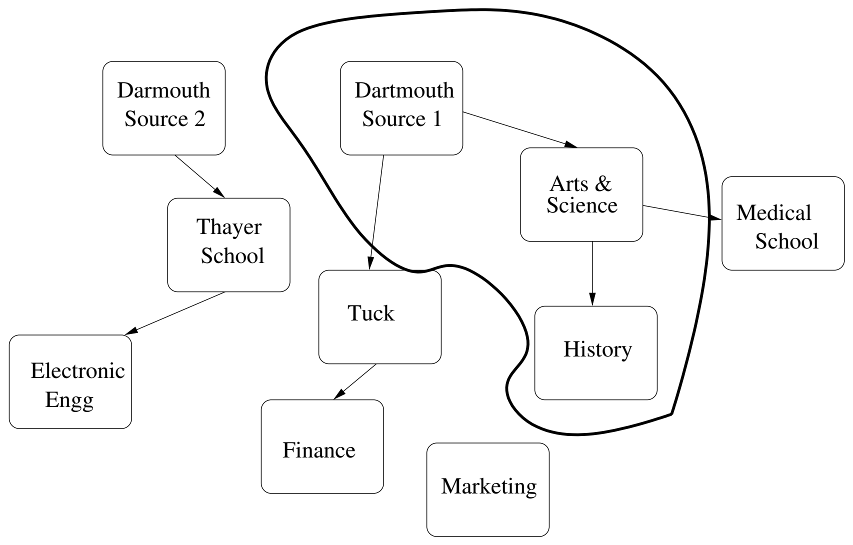


Figure 7: Domain types

that certificate chain is valid and can be used to deduce a resultant ARP. A Source Domain does not have a Predecessor Domain. An example of a Source Domain from Figure 7 is the Domain with the Dartmouth Source key.

2. Leaf Domain: This Domain represents a Domain containing the end user. A leaf Domain does not have a Successor Domain. Usually, Shibboleth users are from Leaf Domains. An example of a Leaf Domain from Figure 7 is the Domain with the History Department public key.

3. Intermediate Domain: This Domain represents an intermediate Domain in a chain of Domains that are connected through delegation. An Intermediate Domain does not usually have users but is used to enforce additional 'filters' or restrictions on a user's ARP. An example of an intermediate domain in figure 7 is the Domain with the Arts and Science public key.

In figure 7, an example of a certificate chain used to derive a resultant ARP for Alice would be

Dartmouth →Arts&Science → History Department → Alice (a History department user)

Domain Controller:

The relation between a Domain and its Domain Controller is 1:1. A Domain can only have one Domain Controller. Each Domain Controller can be associated with only one Domain. The purpose of a Domain Controller is to retrieve all the ARP certificates relevant to a Domain user from this Domain and all the other Domains forming a trust chain after receiving a request from the SPADE Head Controller.

From Figure 6, the SPADE model works as follows:

When a user makes a request to a Shibboleth resource, the Attribute Authority will contact the SPADE Head Controller for the particular user's attributes. The Head Controller will contact the user's Domain Controller and ask for the user's attributes. The Domain Controller retrieves the user's ARP certificate, the Domain Administrator's filter ARP and contacts the domain controller of the 'Predecessor' Domain to retrieve that Domain's ARP relevant to the Shibboleth target's SHAR and URL. Note that since roles are locally defined in each Domain, other Domain's ARPs are selected based on the Shibboleth target's SHAR and URL and not on the user's role.

3.2.2 Head Controller

As can be seen from Figure 8, the Shibboleth Attribute Authority (AA) contacts the SPADE Head Controller which is in contact with all the Domain Controllers in the organization's SPADE infrastructure.

When the Shibboleth Attribute Query Message (AQM) comes into the Attribute Authority (or AA), the AA has to find all releasable attributes associated with the handle presented in the AQM. In the Shibboleth test implementation, this involves consulting the Handle cache (on the origin side) where the Handle was created in the first place, and obtaining the user's name or Identity. Since SPADE is based on SPKI/SDI, all mechanisms must work with this user's public key. This public key can be obtained from an X.509 certificate that the user has used to authenticate to Shibboleth. Thus, we see that X.509 and SPKI/SDSI can both exist together. X.509 can be used for authentication, while SPKI/SDSI can be used for authorization.

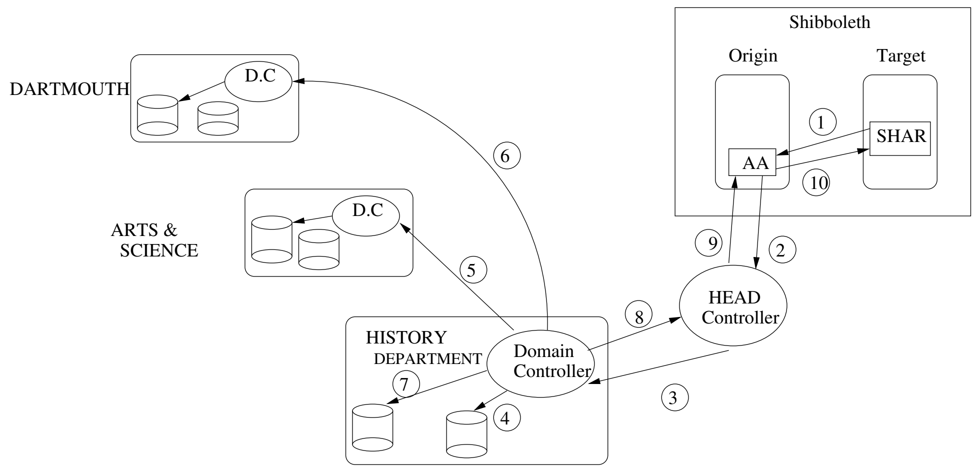


Figure 8: Typical SPADE example

SPADE Flow:

The entire process of receiving an Attribute Query Message and resolving the user's ARP is shown in Figure 8. The steps are described below:

Step 1: The Shibboleth Attribute Requester (or SHAR) at the target site contacts the Attribute Authority (or AA) at the origin site for the relevant user's attributes. The SHAR passes the AA the user handle, and SHAR and URL pair of the target resource that the user wishes to acccess.

Step 2: The AA resolves the handle into the user name and consequently the user's public key (by using an organization-issued X.509 Authentication Certificate from the user's browser) and passes on this information to the the SPADE Head Controller together with the SHAR and URL values requesting for the user's attribute values.

Step 3: The Head Controller resolves the Domain of the user and contacts the user's Domain Controller (or DC). The Head Controller, in effect, hands off the processing to the user's DC and waits for the attribute values. In this example, the user is from the History department.

Step 4: The user's Domain Controller obtains the user's ARP for the particular SHAR and URL pair from the Domain database. It also obtains the Administrator's 'Filter' ARP for this SHAR and URL.

Step 5: The Domain Controller, after checking the Administrator's database for the URL of the 'Predecessor' Domain, contacts the Predecessor's Domain Controller and requests it's Administrators' ARP for the same SHAR and URL. In this case, the History department's Predecessor domain is the Arts and Science Domain.

Step 6: Similary, the user's Domain Controller obtains the Arts and Science Predecessor Domain from it's database which turns out to be the Dartmouth Domain. It retrieves Dartmouth's ARP for the SHAR and URL pair. It also stops retrieving certificates here because it knows (from a list that the Domain Controller has) that Dartmouth is an organization source key.

Step 7: The user's Domain Controller (in the History Department) intersects the user's ARP, the History Department's ARP, the Arts and Science ARP and Darmouth's ARP to form the resulant ARP. The values of the attribute names specified in the resultant ARP are pulled from the user's database. Figure 9 shows the intersection of the relevant SPKI/SDSI certs with ARPs in them.

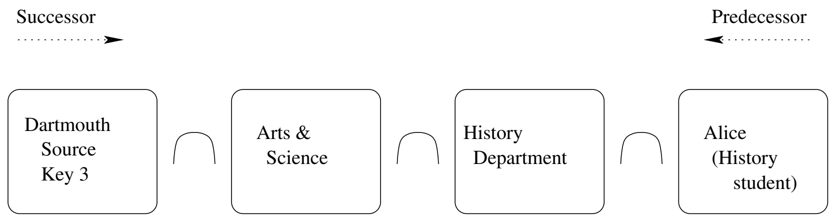


Figure 9: Resolving Attribute Release Policies

Step 8: The user's Domain Controller returns the user's attributes to the SPADE Head Controller.

Step 9: The Head Controller in turn returns the attributes to the AA.

Step 10: The AA, now using the Shibboleth specification, bundles the attributes into SAML and sends them off to the SHAR which decides whether the user gets access to the target resource or not.

3.3 SPADE Implementation

I have implemented a web-based GUI that allows the Domain users and Administrator to log on and manage their policies. For the administrator, this involves authorizing new users in the system, creating default attribute release policies, and creating roles for the different users to be assigned to. A user can use the GUI to create and modify his attribute release policy.

Administrator functions:

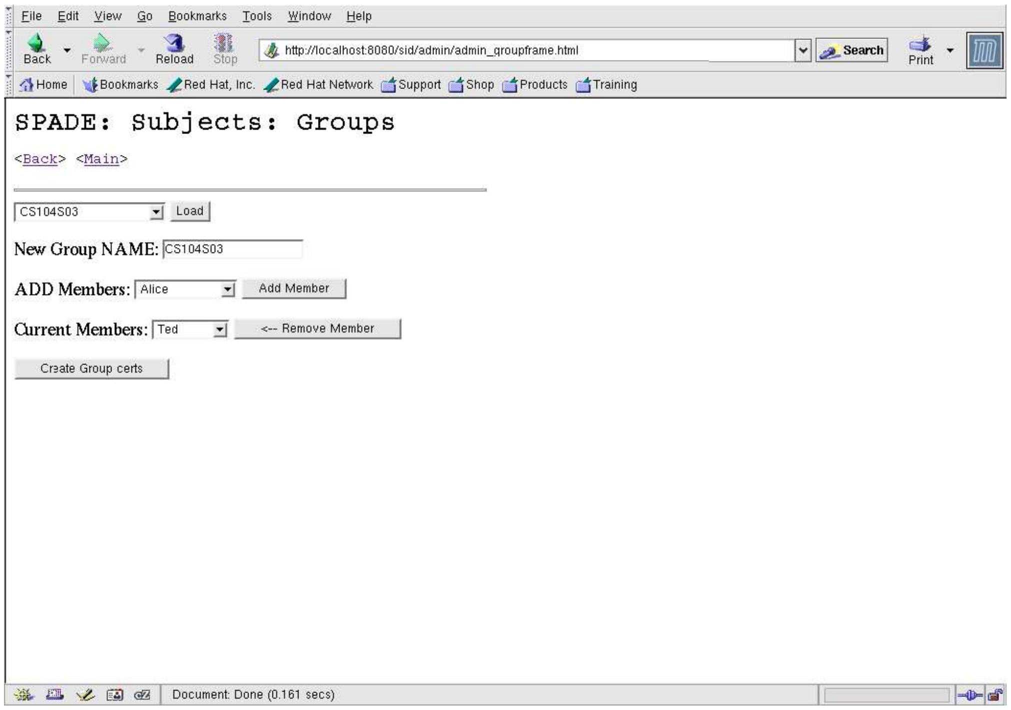
1. Binding Names to Public Keys: In order to work with public keys, the human Administrator needs to bind them to local SDSI names using a SPKI/SDSI Name certificate.

2. Creating Roles: Administrators create roles by specifying which users to add to a new or pre-existing group. Note that all user names specified in the GUI are taken from name certificates that the administrator has created using the 'Create Name Certificate' menu.

3. Attribute Release Policies:

    (a) Types: Administrators have to create two types of Attribute Release Policies:

        i. Filter ARP: This is the attribute release policies that is used when a user has not created one for a specified SHAR and URL. This ARP is also used



http://localhost:8080/sid/admin/admin_groupframe.html

SPADE: Subjects: Groups

<Back> <Main>

CS104S03

New Group NAME: CS104S03

ADD Members: Alice

Current Members: Ted

Document: Done (0.161 secs)

Creating Roles or Groups

```
(cert
 (issuer
  (name
   (public-key
    (rsa-pkcs1-md5
     (e #010001#)
     (n
      |AMh8A5+uINnrESIHlW2Gk1Uf4kthKnWnjNyg8wXNJXPaEXax/DPTNTMDuUC
      ZRVyFGohuqr87VAR4ZFU99OkIWeryPwKsXjwyHr7Bq1K61+n0HByRNhZkBFj
      qajpreByRw9V+fwV9O8WPIVkMvIgTpcMF6rCKfU3f5M2OROpJ1ZVZ|)))
   CS104S03))
 (subject
  (public-key
   (rsa-pkcs1-md5
    (e #010001#)
    (n
     |ANbQJy7L3s9PDHippZNcLPyh58WEWxXY30rgg4Y5cs0vnpx6LEgaK+c+VC+PS
     t48cY7pZdSGXa+fZp162cbqsLM0nBSAMkJ1VCYP/3v91vuS9hWY43iPy1PV57p
     cOiibValFpTUmO/BIeKR9do41nW9hGX8fJv8Y3enZ5cLupq11|))))
  (valid
   (not-before "2003-01-01_00:00:00")
   (not-after "2004-01-01_00:00:00")))
```

Figure 10: Example of SPKI/SDSI Name Certificate

as a filter for the ARP of a role in the domain. For example, this ARP can specify that all users who have the role ’Student’ cannot release their Credit Card information and will be filtered through this ARP. If a user has not specified an ARP, then this ARP will be used where the credit card information will naturally not be released.

ii. Releasable Attributes ARP: This policy specifies all the attributes that the user can ’work’ with when creating his ARP. It works as a template for a user to create his ARPs with. For example, a user may want to release his Name, Address and Department Affiliation to a Shibboleth target site. The user can do so only if these attributes are in the Releasable Attributes ARP.

An example of an ARP embedded in SPKI/SDSI is shown in Figure 11.

4. Design: Attribute Release Policies are created in two stages:

(a) SHAR and URL address are paired together. This includes wild-card entries. This is shown in Figure 13.

(b) These pairs are then associated with roles and attributes. One pair may be associated with any number of roles. The ARP is then created.

5. Hidden attributes: Administrators are also able to embed attributes into their Filter ARP which the user cannot see when creating his individual ARP. This is so that attributes from out-of-band agreements between the user’s organization and the target site may be appropriately conveyed while making it transparent to the user. The organization can choose to inform the user about this underlying attribute flow.

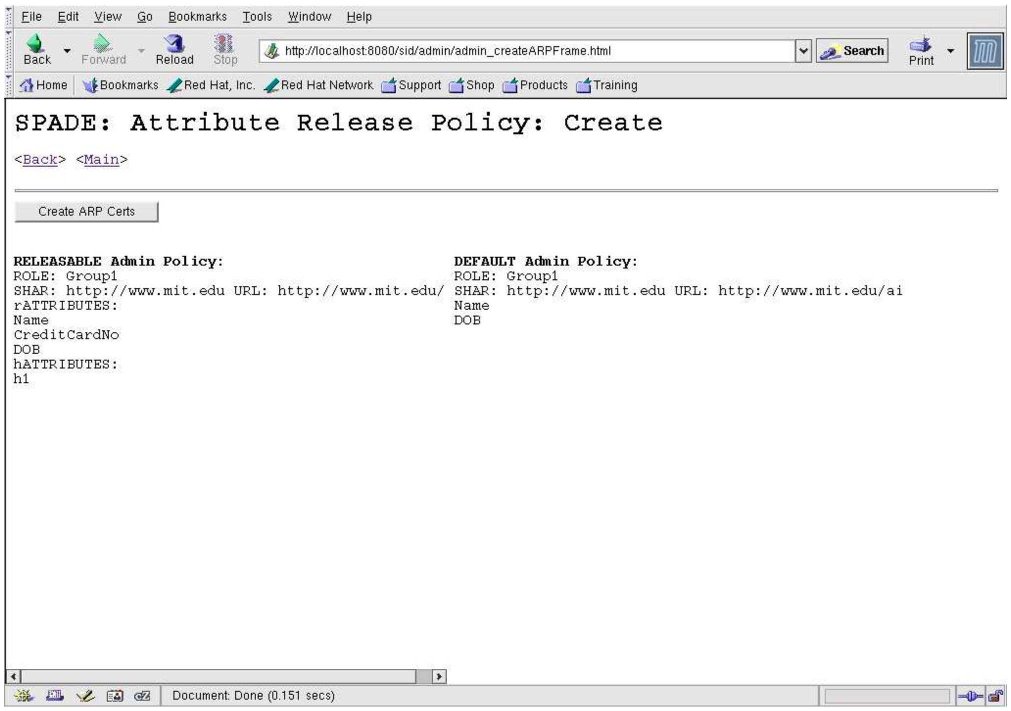
User functions:

1. The user uses the SPADE GUI to create his Attribute Release Policies. This entails two steps:

   (a) The user chooses which of his Roles he wants to create an ARP for. He then,

```
(cert
 (issuer
  (public-key
   (rsa-pkcs1-md5
    (e #010001#)
    (n
     |AMh8A5+uINnrESIHlW2Gk1Uf4kthKnWnjNyg8wXNJXPaEXax/DPTNTMDuUCZR
     VyFGohuqr87VAR4ZFU99OkIWeryPwKsXjwyHr7Bq1K61+n0HByRNhZkBFjqajp
     reByRw9V+fwV9O8WPIVkMvIgTpcMF6rCKfU3f5M2OROpJ1ZVZ|))))
  (subject
   (name
    (public-key
     (rsa-pkcs1-md5
      (e #010001#)
      (n
       |AMh8A5+uINnrESIHlW2Gk1Uf4kthKnWnjNyg8wXNJXPaEXax/DPTNTMDuUC
       ZRVyFGohuqr87VAR4ZFU99OkIWeryPwKsXjwyHr7Bq1K61+n0HByRNhZkBFj
       qajpreByRw9V+fwV9O8WPIVkMvIgTpcMF6rCKfU3f5M2OROpJ1ZVZ|)))
   CS104S03))
 (propagate)
 (tag
  (adminARP
   (ROLE (CS104S03)
     (SHAR (http://www.dartmouth.edu)
      (URL (http://www.dartmouth.edu/cs)
      (ATTRIBUTES (CreditCardNo)))))
   (ROLE (CS104S03)
    (SHAR (http://www.amazon.com)
     (URL (http://www.amazon.com/books)
      (rATTRIBUTES (Address CreditCardNo DOB Email)))))))
 (valid
  (not-before "2003-01-01_00:00:00")
  (not-after "2004-01-01_00:00:00")))
```

Figure 11: Example of SPADE Authorization Certificate with ARP delegating to Role CS104S03


File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://localhost:8080/sid/admin/admin_createARPFrame.html
Search Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training
Create ARP Certs
Document: Done (0.151 secs)

SPADE: Attribute Release Policy: Create

<Back> <Main>

```
RELEASABLE Admin Policy:                          DEFAULT Admin Policy:
ROLE: Group1                                      ROLE: Group1
SHAR: http://www.mit.edu URL: http://www.mit.edu/ SHAR: http://www.mit.edu URL: http://www.mit.edu/ai
rATTRIBUTES:                                      Name
Name                                              DOB
CreditCardNo
DOB
hATTRIBUTES:
h1
```

Figure 12: Attribute Release Policy Types

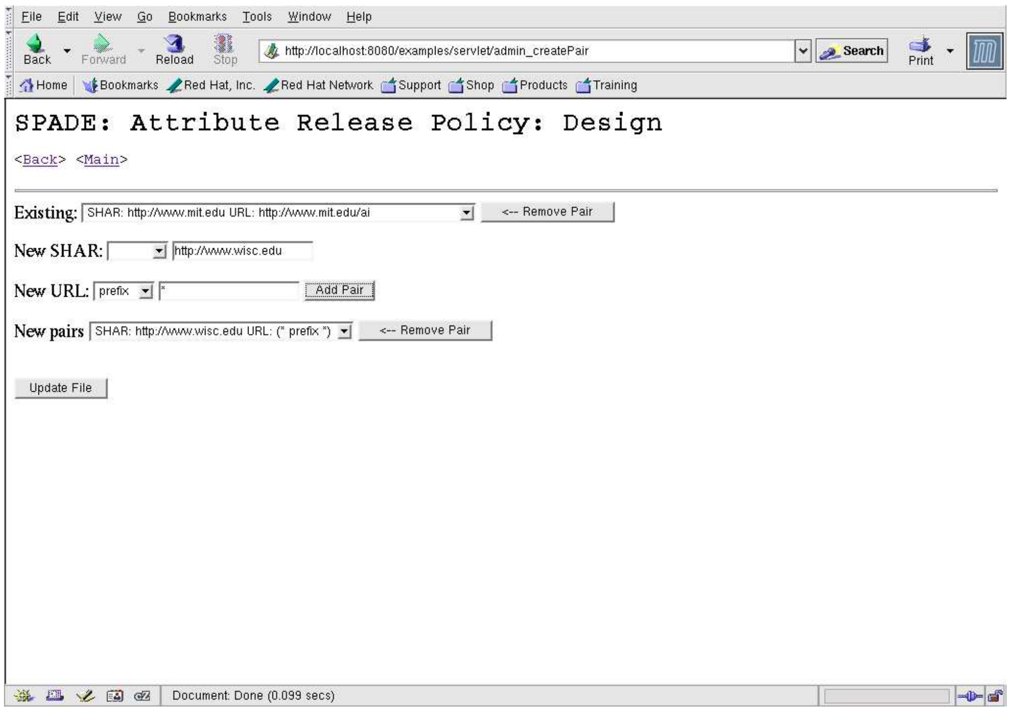
selects which attributes he wants to release for a particular SHAR-URL pairing for that role.

Policies:

Specification:Attribute Release Policies are specified using SPKI/SDSI tags which are part of the standard [Ellison2]. These are embedded as one of the fields in a SPKI/SDSI authorization certificate.

Dissemination: Policies are created by users in their respective domains. The trust relationship establishment between domain Administrators makes sure that all required policies may be retrieved. Policies are thus retrieved rather than disseminated. In SPADE, the Policy Enforcement Point is (logically) the Head controller,while the Policy Retrieval and Policy Decision takes place at the user's Domain Controller. In effect, the Head controller never sees the policy but only works with the user's attribute values that are returned from his

Domain.


File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://localhost:8080/examples/servlet/admin_createPair
Search Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training

SPADE: Attribute Release Policy: Design

<Back> <Main>

Existing: SHAR: http://www.mit.edu URL: http://www.mit.edu/ai <-- Remove Pair

New SHAR: http://www.wisc.edu

New URL: prefix * Add Pair

New pairs SHAR: http://www.wisc.edu URL: (" prefix ") <-- Remove Pair

Update File

Document: Done (0.099 secs)

Figure 13: SHAR and URL pairings

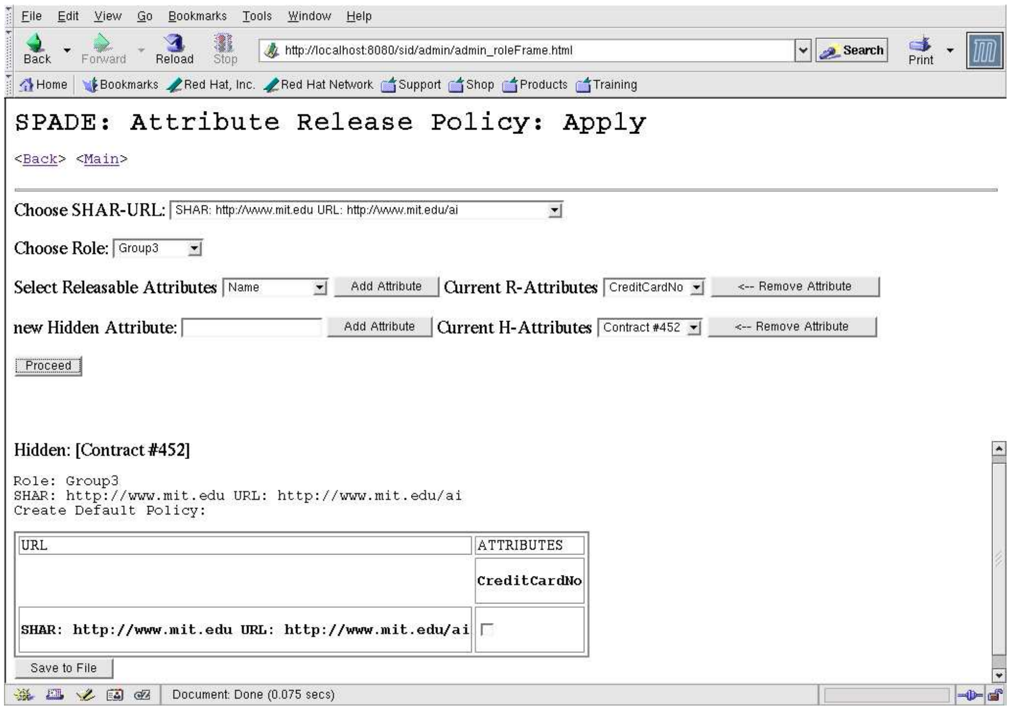
3.4 SPADE Summary

Thus, SPADE looks at a number of policy related and security management issues, as enumerated below:

1. SPADE uses SPKI / SDSI certificates to delegate authority and to manage policies in an organization.

2. SPADE allows user to define and create their own ARPs, specific to their organizational assigned role in that Domain.

3. SPADE allows other Domains in the organization to exert their influence on the individual user's ARP and to thus, base the final ARP on the intersetion of these ARPs.


File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://localhost:8080/sid/admin/admin_roleFrame.html
Search Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training

SPADE: Attribute Release Policy: Apply

<Back> <Main>

Choose SHAR-URL: SHAR: http://www.mit.edu URL: http://www.mit.edu/ai

Choose Role: Group3

Select Releasable Attributes Name | Add Attribute | Current R-Attributes CreditCardNo | <-- Remove Attribute

new Hidden Attribute: | Add Attribute | Current H-Attributes Contract #452 | <-- Remove Attribute

Proceed

Hidden: [Contract #452]

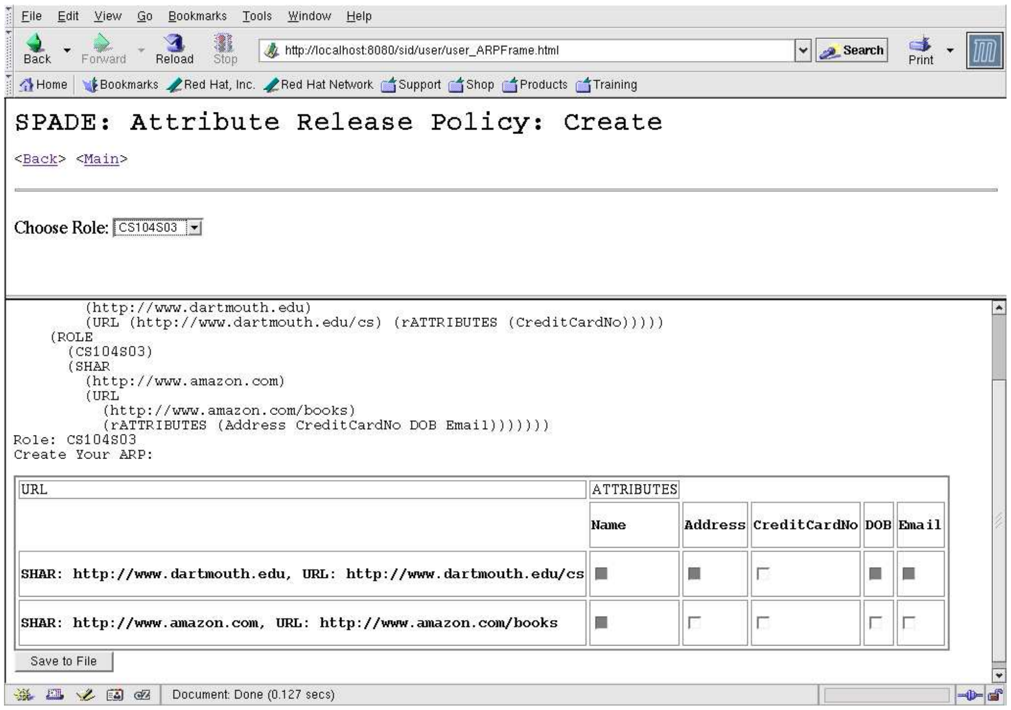
```
Role: Group3
SHAR: http://www.mit.edu URL: http://www.mit.edu/ai
Create Default Policy:
```

URL	ATTRIBUTES
	CreditCardNo
SHAR: http://www.mit.edu URL: http://www.mit.edu/ai	☐

Save to File

Document: Done (0.075 secs)

Figure 14: Associating SHAR-URL pairings with Roles


File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
http://localhost:8080/sid/user/user_ARPFrame.html
Search
Print
Home Bookmarks Red Hat, Inc. Red Hat Network Support Shop Products Training
Save to File
Document: Done (0.127 secs)

SPADE: Attribute Release Policy: Create

<Back> <Main>

Choose Role: CS104S03

```
        (http://www.dartmouth.edu)
        (URL (http://www.dartmouth.edu/cs) (rATTRIBUTES (CreditCardNo)))))
    (ROLE
      (CS104S03)
      (SHAR
        (http://www.amazon.com)
        (URL
          (http://www.amazon.com/books)
          (rATTRIBUTES (Address CreditCardNo DOB Email)))))))
Role: CS104S03
Create Your ARP:
```

URL	ATTRIBUTES				
	Name	**Address**	**CreditCardNo**	**DOB**	**Email**
SHAR: http://www.dartmouth.edu, URL: http://www.dartmouth.edu/cs	■	■	□	■	■
SHAR: http://www.amazon.com, URL: http://www.amazon.com/books	■	□	□	□	□

Figure 15: SPADE User page to create ARP

4 Related Work

In this Chapter, I describe some of the work that is relevant to my research. I concentrate on XML based policy and *Trust Management* systems. I also describe their relation with SPKI / SDSI and SPADE,where appropriate.

4.1 OPS

Open Profiling Standard (or *OPS*) [Hensley2] is very similar to the Shibboleth ARP model. OPS allows users that access online resources through web browsers the power and flexibility to specify what attributes they release to a particular website. This is done using a "Personal Profile".

Like Shibboleth, OPS has three main principles that are documented in [Hensley2]. They are as follows:

1. "**Control by Source**" : According to this, the owner of the attributes data is the entity that controls access to it. Similarly, in Shibboleth, a user is in control of access to his attributes via his ARP.

2. "**Informed Consent**" : This principle states that an entity requesting user attributes must have the user's consent. In the case of Shibboleth, this is again done through the user's ARP.

3. "**Value Exchange"** : According to this principle, there must be a give and take relationship. i.e. an entity cannot make a request and obtain a user's attributes without providing something in exchange. This is usually access to a resource that the asking entity is in control of.

OPS is a standard proposed mainly for interaction between consumers and online e-commerce sites. Users are meant to create their personal profiles using OPS and store them on their computers. When a user contacts an online site such as Amazon, for example, Amazon can check the user's profile to obtain the user's preferences for music, books and toys for example. The user has the power to specify exactly what gets released to Amazon or any other site.

An important difference with the Shibboleth ARP model is the OPS allows the target site (i.e. the online site holding the resource) to write to the user's profile. This is so that information stored about a user that provides greater convenience and value for future interactions can be stored in the user's Personal Profile and used on the user's next visit. Of course, the user is given the power to decide whether his Personal Profile can be written to or not.

Another important difference with Shibboleth, and a disadvantage of OPS. is that it needs modification to the web browser. This is so that the user's Personal Profile can be exchanged over HTTP (as mentioned in [Hensley1]) and can be written to by the online site. Two methods of implementing OPS are by using "Java Applets with trusted code" or by using a browser plug-in.

OPS was proposed in 1997 and may have contributed to some of the ideas in Shibboleth, especially with request to attribute exchange. It is important to note that a Personal Profile is not a cookie. Also, the OPS standard does not recommend using a Personal Profile to store important information such as a credit card number. Instead, a Personal Profile may be used to interact with an online '*Wallet*' which can store other important information about a user.

4.2 XML-Based Techniques relevant to Privacy

4.2.1 P3P

The Platform for Privacy Preferences Project (or P3P) [Cranor] is a protocol developed by the World Wide Web Consortium (or W3C). According to [Cranor], *"It provides a way for a Web site to encode its data collection and data-use practices in a machine-readable XML format known as a P3P policy."*

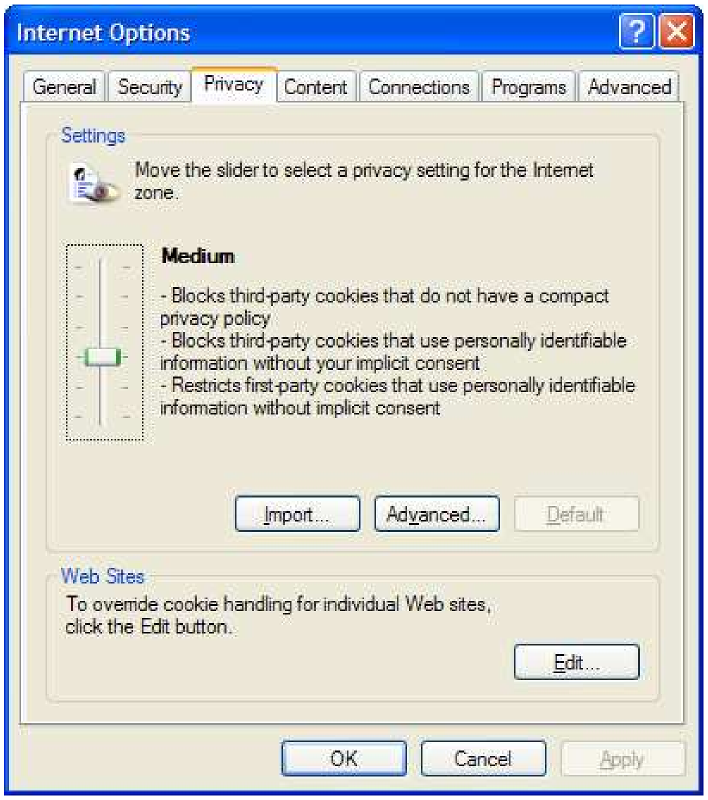
The main goal of P3P is to enable users and machines reading the data collection policy of a Web site to be able to deduce exactly what it means. It does so by making use of the semantics of XML tags. This standard was proposed because numerous Web sites currently have data collection policies (such as what cookies they use and what user information they store) that are specified in plain English and are too convoluted and ambiguous to understand.

Microsoft's Internet Explorer 6 has support for P3P, and provides a GUI by which a user can specify their privacy preferences. Current support only involves cookies. This is shown in the figures 16 (taken from [Madsen2]). According to this, if a user visits a Web site which uses cookies in an inconsistent manner with the user's privacy policy (specified using P3P) then the user is warned by the Internet Explorer status bar, shown in figure 17 (taken from [Madsen2]).

4.2.2 XACML

eXtensible Access Control Markup Language (or *XACML*) [Godik] is an XML based language for expressing a user's security policies and involves authorization for resources. Its main features are it support for fine grained authorization control. XACML provides a means of specifying a Shibboleth ARP.

The three main elements in XACML, according to [Godik] are the <Rule>, <Policy> and <PolicySet> elements. <Rules> are boolean expressions that do not convey authorization by themselves. A <Policy> element consists of a set of <Rules> and a procedure for combining the values of the <Rules> into an authorization decision. A <PolicySet> element contains multiple <Policy> or <PolicySet> elements and a procedure for combining the results of their evaluation. Authorization decisions can be made based on <Policy> or



Move the slider to select a privacy setting for the Internet zone.

Medium

- Blocks third-party cookies that do not have a compact privacy policy
- Blocks third-party cookies that use personally identifiable information without your implicit consent
- Restricts first-party cookies that use personally identifiable information without implicit consent

To override cookie handling for individual Web sites, click the Edit button.

Figure 16: P3P support in Internet Explorer 6

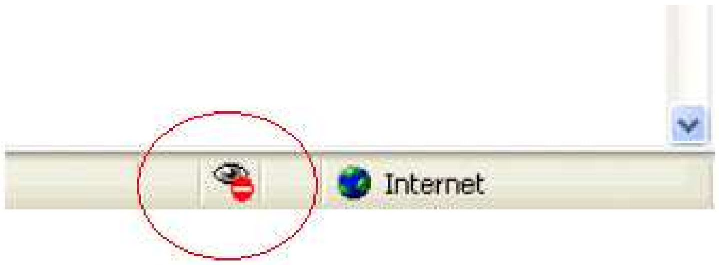


Figure 17: P3P support in Internet Explorer 6

<PolicySet> elements.

XACML also supports Roled Based Access Control (or RBAC). A policy may contain the role or set of roles that are allowed to perform a particular action.

XACML is only a language for specifying policies and how to combine them to deduce authorization. The standard does not specify or provide support for defining a policy based infrastructure. Policy dissemniation, retrieval, enforcement and administration are also not specified as part of the specification.

4.2.3 SAML

Security Assertion Markup Language (or SAML) [Maler] is another OASIS initiative like XACML. SAML is XML-based and provides a standard way to define information about user authentication, authorization and attributes between online sites. SAML was mainly proposed to allow Web site to be able to exchange user information in a compatible manner without needing to change their internal (possibly proprietary) access control mechanisms.

'*Single Sign On*' (or *SSO*) between Web sites is an important application of SAML. Using SSO, users can log into a Web site, provide their attribute information and then log into another Web site in the same session while the first Web site sends the second Web site that user's attributes in a SAML format. Thus, the user does not have to provide his information again.

The main difference between SAML and XACML is the function of both standards. SAML is used to specify and carry information between web sites. It makes no effort at privacy or policy specification. XACML, on the other hand, is used to combine attributes to form and enforce policies. An example of both standards working together would be in Shibboleth [ShibDraft] where SAML is used to carry user attributes between the origin site and the target site (part of the Shibboleth specification), and XACML would be working at the sending and receiving end of the channel to relate those attributes to policies and enforce policies (not part of the Shibboleth specification).

4.2.4 IBM EPAL

Enterprise Privacy Authorization Language (or *EPAL*) [Ashley] is an XML-based model for *"formalizing enterprise-internal privacy policies...EPAL formalizes privacy authorization for actual enforcement within an enterprise or for business-to-business privacy control."*

While EPAL is relevant and provides some of the functionality of policy specification that SPADE provides, it has a few differences:

1. *"EPAL does not conform to any assumptions, data structures or features of a pre-existing product or tool."* This may not be desirable at a low-level when working with an organizations own policy specification mechanism. SPADE can easily convert policies to S-expressions and back.

2. According to [Ashley] EPAL was not designed for, *"Manipulation of EPAL by data subjects to set their preferences"*. This implies that, unlike SPADE, domains using EPAL cannot manipulate policy according to their discretion. There must be a management policy or entity in the system which they can subscribe to to change individual's and the organization's privacy policy.

4.3 Trust Management

According to [Vollbrecht],

> *" ' Trust ' is necessary to allow each entity to 'know' that the policy it is authorizing is correct".*

The concept of Trust Management, introduced by Blaze et al. [Blaze1, Blaze2] is *"a unified approach to specifying and interpreting security policies, credentials, and relationships that allows direct authorization of security-critical actions".*

Trust management systems handle security policies, authentication as well as authorization. These are usually specified by binding a public key to a set of attributes.Trust Management is a powerful approach to handling distributed security policies and decentralization, as it can handle delegation and policy specification at different hierarchical layers.

According to [Blaze4], a Trust Management system must have five basic components:

" 1. A language for describing 'actions', which are operations with security consequences that are to be controlled by the system.

2. A mechanism for identifying 'principals', which are entities that can be authorized to perform actions.

3. A language for specifying application 'policies', which govern the actions that principals are authorized to perform.

4. A language for specifying 'credentials', which allow principals to delegate authorization to other principals.

5. A 'compliance checker', which provides a service to applications for determining how an action requested by principals should be handled, given a policy and a set of credentials."

An integral and central part of a Trust Management system is the compliance checker. A request, a policy and a set of credentials (usually a public key certificate) are input to compliance checker which answers a "yes" or "no", depending on whether the credentials and request complies with the policy.

Policy Maker [Blaze2] and KeyNote [Blaze4] are two examples of Trust Management systems.

4.3.1 PolicyMaker

PolicyMaker is a Trust Management system developed by Matt Blaze et al [Blaze2] . The main characteristics of PolicyMaker are:

1. It has a common language for specifying policies, credentials and trust relationships.
2. It supports complex trust relationships as well as a local control of them that prevents the need for a hierarchy of Certificate Authorites (such as in X.509).
3. The process of the verification of the credentials is independent of the credentials semantics or syntax themselves.
4. Binding of a public-key to attributes or policy. Thus, it supports anonymity as the identity of the individual is not necessary to decide on the authorization.
5. "Trust may be deferred". This means that one party may allow another party to issue credentials for itself. This is like delegation.
6. PolicyMaker is built as a "Trust Management layer" that networks can use.

4.3.1.1 PolicyMaker language The PolicyMaker language can express authorization policies and trust deference policies through assertions. These assertions can be written in any 'safe' language such as Java, safe Tcl and awk.

4.3.2 KeyNote

KeyNote [Blaze4] was designed according to the same principles as PolicyMaker, using credentials that directly authorize based on a public key. Two additional design goals for KeyNote were standardization and aiding the ease of integration into applications.

To address these goals, KeyNote assigns more responsibility to the trust management engine that PolicyMaker does and less to the calling application. KeyNote also requires that credentials and policies be written in a specific assertion language to integrate well with KeyNote's compliance checker.

Relation to SPKI/SDSI: According to Matt Blaze et al [Blaze2], SPKI is similar to the KeyNote trust management belief that "certificates can be used directly for authorization rather that simply for authentication".

4.3.3 Distributed Trust Management System

The Distributed Trust Management System at the University of Maryland, Baltimore County [Kagal] uses rights and delegations with certificates to create a trust management system. The main characteristics of this system are:

1. It posseses a method for access control across domains that handles complex inter-domain trust relationships. A main 'claim to fame' of this system is that it allows agents to delegate any right that they may have, provided they have the right to delegate themselves (like in SPKI/SDSI).

2. Architecture: Entities in the system are identified by their X.509 Identification Certificate Each group of agents is protected by special 'security' agents that authorize access for regular agents in their groups and trust other security agents. All communication is done via signed messages called Signed Message Objects (or SMOs).

3. Trust information and policies are expressed in Prolog and can model permissions and delegations. A security agent stores the organization's policy in a Prolog knowledge base. When it receives a *SMO* (*Signed Message Object*) it verifies it by inserting the Prolog statement into the knowledge base.

4. The infrastructure uses X.509 certificates and Prolog policies to enforce security. Authentication is done using X.509 Identification certificates. Authorization is done using X.509 Attribute Certificates containing signed Prolog statements.

5. Authorization is done through a chain of trust using rights and delegations.According to Lalana Kagal et al [Kagal], "Permissions are extended by delegation from an authorized agent".

6. According to Lalana Kagal et al[Kagal], *"Obligations, entitlements and prohibitions are not modeled as of yet, but will be done."*

4.4 Trust Establishment Framework

4.4.1 IBM Trust Establishment Framework

According to [Herzberg], the function of the IBM Trust Establishment Framework (or TEF) is to map the subject of a certificate to a role based on the certificate and policy. It is not meant to authenticate. The main characteristics of the IBM TEF are:

1. The IBM TEF supports (according to [Herzberg])

> *"complex networks of trust rather than requiring a pre-defined tree with a fixed 'root certification authority. There is a bottom-up, "grass roots" buildup of trust and the public key infrastructure, mirroring the real world."*

The TEF's issuer and subjects are both public keys that are stored in a generic system certicate object and retirieved from an X.509 Attribute Certificate using two special fields called 'issuerAltName' and 'subjectAltName'.

2. According to [Herzberg], *"The goal of the IBM TEF is not to replace existing access control systems, but to extend them by mapping unknown users to roles."*

IBM Trust Establishment Framework defines a Trust Policy Language (or TPL) for specifying policies.

4.4.2 Trust Policy Language (TPL)

The main characteristics of TPL are:

1. TPL is an XML based policy language where flexible policies may be used to assign roles to entities in the system. This is mainly to ensure compatability with other system with which a TPL policy may be exchanged.

2. TPL is non-monotone.

3. TPL can support delegation (with a 'repeat' tag) and thresholds.

Some of the advantages of IBM TEF over Trust Management systems are:

1. Trust Management systems provide both Trust Establishment as well as Access Control. This results in a more complex system and makes it difficuilt to integrate with existing systems, especially if only an authorization mechanism is required.

2. Trust Management systems like PolicyMaker and KeyNote possess complex programming languages that is hard for non-programmers to use. The IBM TEF uses the XML based Trust Policy Language.

3. IBM TEF is assertion non-monotonic in that it provides positive as well as negative certificates, unlike PolicyMaker and KeyNote.

Relation to SPKI/SDSI: The IBM TEF uses a public key as a subject's global name, just like SPKI/SDSI.

Drawback of IBM TEF: According to Lalana Kagal et al. [Kagal], the IBM Trust Establishment Framework suffers from a lack of flexibility since it only considers RBAC.

4.5 Privilege Management Infrastructure (PMI)

According to [Chadwick1], a *Privilege Management Infrastructure* stores authorization information (such as a user's privileges or attributes) rather than authentication information. David Chadwick says,

“ A PMI is to authorization what a PKI is to authentication”

Table 2 gives a comparison of PKIs and PMIs (as taken from [Chadwick2])

Concept	PKI entity	PMI entity
Certificate	Public Key Certificate (PKC)	Attribute Certicate (AC)
Certificate issuer	Certification Authority (CA)	Attribute Authority (AA)
Certificate user	Subject	Holder
Certificate binding	Subject’s Name to Public Key	Holder’s Name to Privilege Attribute(s)
Revocation	Certificate Revocation Lists (CRL)	Attribute Certificate Revocation Lists (ACRL)
Root of trust	Root Certification Authority or Trust Anchor	Source of Authority
Subordinate authority	Subordinate Certification Authority	Attribute Authority (AA)

Table 2: Comparison of PKIs and PMIs

Some of the differences between PKI and PMI are as follows:

1. Certificate: In PKI, an authentication certificate is used and binds a user’s name to a public key. In PMI, an Attribute Certificate is used for authorization which binds a user’s name to a set of attributes. Note that PMIs are usually X.509 based.
2. Signer: While *Certification Authorities* (or *CA*) sign certificates in PKI, *Attribute Authorities* (or *AA*) sign certificates in PMI. Note that these are not the same as the AAs in Shibboleth.
3. Root of trust: A root CA heads as the root of trust in a PKI. In a PMI, this is a *Source of Authority* (or *SOA*). A SOA has correspondingly similar powers to a CA like delegating authority to a subordinate AA.
4. Revocation: CAs revoke certificates using Certificate Revocation Lists (or CRLs). An AA issues a corresponding Attribute Certificate Revocation List (or ACRL).

4.5.1 PERMIS

PERMIS [Chadwick1] is an example of X.509 based PMI. But, PERMIS also authenticates and is thus, also defined as a Trust Management System. Some of the main characteristics of PERMIS are as follows:

1. PERMIS has both access control and authorization policy components.

2. The PERMIS architecture has two subsystems - 'privilege allocation' subsystem to allocate privileges to users and a 'privilege verification' subsystem used to verify them later. As mentioned in [Chadwick1] , *"The SOA uses the privilege allocation subsystem to issue X.509 role based attribute certificates to the users"*. The 'privilege verification' subsystem also performs authentication.

3. PERMIS uses an XML based policy language with a hierarchical RBAC model. In PERMIS, privileges given to a role are automatically inherited by the roles 'superior' to it.

4. Since PERMIS is a Trust-Management system, it must have five basic components (according to [Blaze4]). These are:

   (a) A language for describing "*actions*", which is XML and DTD based in PERMIS.

   (b) A mechanism for describing "*principals*", which perform the actions. This is done in PERMIS using global names from X.509 Attribute Certificates (or ACs).

   (c) A language for specifying "*policies*", which apply to the actions that the principals perform. This is also done in the PERMIS XML based policy language.

   (d) A language for specifying "*credentials*", which facilitates delegation of authorization. This is done in PERMIS using X.509 ACs.

   (e) A "*compliance checker*", which determines how an action requested by a principal should be handled given a policy and a set of credentials.

5. PERMIS stores policies and X.509 ACs in an LDAP so that the privileges may be widely distributed over the Internet.

4.6 Policy Models

PolicyMaker, KeyNote and PERMIS are trust management systems that include a policy model. However, there are also Policy models proposed without the other features of a Trust Management System. Two such examples are Ponder [Dulay] and SPOCP [Hedberg2, Hedberg2].

4.6.1 Ponder

Some of the characeristics of Ponder (from [Dulay]) are as follows:

1. Ponder is an object-oriented language that can specify both security policies and management policies. A security policy specifies what the reply to a query or action will be. A management policy specifies how that query or action is handled to arrive at that reply.

2. Ponder has Role Based Access Control (or RBAC).

3. The model defined objects for policies, for domains and for policy enforcement agents and outlines the interactions between them. A policy may apply to a domain object or a set of domain objects.

4. This model also proposes support for Ponder, which according to [Dulay] is *"a new policy language that combines structuring ideas from object-oriented languages with a common set of policy basic types"*.

5. Ponder is more concerned with how policies are disseminated to entities that will interpret them, than with how policy will be specified.

4.6.2 SPOCP

Simple Policy Control Protocol (or SPOCP) [Hedberg4, Hedberg2] is used for general authorization. SPOCP can be thought of as a policy engine that tests if an authorization request from a client ought to be allowed or not, given a set of policies. Some of the main characteristics of SPOCP are:

1. The SPOCP Authorization server is designed around the Policy Engine. The Policy Engine answers ’yes’ or ’no’ to a ’Query’ from a client: "Does a potential action comply with or violate a particular policy?". Servers respond “+OK”, “-NO” or “-ERR” to a query request.

2. Rules are monotonic in that they can only grant something, but not make negative statements. When a query is posed to the system, the system searches for a rule that permits the queried action. If one such rule is found the system will stop there and return its decison. It will not collect all the rules that may relates to an action, although more may be present.

3. SPOCP uses S-expression to specify queries and policies, just like SPKI/SDSI. The policy engine only compares (or intersects) S-expressions. It does not concern itself with the semantics of the policy.

According to Roland Hedberg et al in [Hedberg4], SPOCP chooses S-expressions because the developers felt the need to have a policy syntax that is independent of the application the processes it. In this way, multiple systems and organizations can use the system without being constrained to using a fix set of processing tools, such as XML for example.

Table 3: Comparison of Policy Management Systems

Policy Scheme	SPKI/SDSI	XACML	PolicyMaker, KeyNote	Ponder	PERMIS	IBM TPL	ACL	DTM, Maryland
Trust Management System?	No	No	Yes	No	Yes	No	No	Yes
Summary	PKI standard that is public-key centric. Used mainly for lightweight authorization. Uses S-expressions to express policies in certificate ’Tags’.	Language to express Authorization Policies with <Rule>, <Policy> and <PolicySet> elements	Trust Management System with language to specify Policies	OO Language for specifying security policies	A PMI. Uses X.509 ACs.	Trust Establishment Framework. Authorization, but no Authentication	Username and Password linked with Individual Attributes. One-to-one-relationship.	Handles complex inter domain trust relationships.Claim to fame is allowing users to delegate any right they have.
RBAC	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Principal	Public Key	<Subject> Usually text	Public key bound to attributes	Object or set of Objects	Identity in X.509 AC	Public key in modified X.509 AC	Username	Identity in X.509 AC
Certificate based?	Yes	No	Yes	No	Yes	Yes	No	Yes
Language	S-expressions	XML	PolicyMaker	Ponder	XML	XML	None	Prolog
Assertions	Monotone	Non-Monotonic	Monotonic	Non-Monotonic	Monotonic	Non-Monotonic	Monotonic	Monotonic
Delegation?	Yes	No	Yes	Yes	Yes	Yes	No	Yes
Drawbacks	Not as flexible as trust management systems. Tags that carry authorization information are application specific. No provision to specify depth of delegation.	Only policy syntax support. Does not specify infrastructure or considers Trust issues.	Complex semantics. Difficult for non-programmers to work with. CRLS are not built in, and signature verification is not part of the system.	Complex. Too inefficient for minimalistic policies.	Inherits the flaws of X.509 Attribute Certificates.	Not as flexible as other systems. Only considers RBAC. No support for Authentication. Does not address policy lifecycle management.	Not flexible or scalable. No support for delegation or Distributed Trust Management	Still a prototype system. Obligations, entitlements and prohibitions are not modeled as of yet.

5 Summary

I have designed and developed a Trust Establishment system using Role Based Authorization for creating minimalistic policies. These policies enforce privacy constraints on user thats access the Shibboleth system.

Part of my argument for using SPKI/SDSI is that for a Distributed Trust Management system working with minimalistic policies (such as Attribute Release Policies for Shibboleth), SPKI/SDSI is a good fit. As David Chadwick said [Chadwick3], *"it is simple (two 'Simples' in SPKI/SDSI !) and easy to understand, process and code applications for."* A disadvantage is that it may suffer from a lack of flexibility that distributed trust and policy management systems offer such as PERMIS, Ponder, PolicyMaker, and Keynote. Also, it is also not as flexible as the XML solution to policy management which is XACML. However, in certain 'simple' cases such as these, SPKI/SDSI is validated. Moreover, a SPKI/SDSI certificate (with it's signature) is portable while still maintaining its assertion. And lastly, in the case of Shibboleth the SPKI/SDSI tags, which are part of the standard, aid in ARP creation with their wild-cards and other special properties.

Other disadvantages of SPADE are given below:

1. According to Matt Blaze et al [Blaze2], SPKI certificates lack flexibility as they have a limited number of data fields. However, this may not be so bad as the goal of the SPADE system is to choose simplicity over excessive flexibility.

2. In SPADE, an Authorization certificate "Tag" field is used to store the attributes or policy of the issuer. The developers of the IBM Trust Establishment Framework [Herzberg] object to this approach, as the issuer of the certificate and the resource owner may be different entities. Hence, there might be a policy compatibility problem. In the TE framework, the data in the certificate is separated from the policy itself.

In conclusion, this work has explored the mechanism of using SPKI/SDSI to store user's policies and the using a public key infrastructure to retrieve, intersect and derive new policies. While this body work does not claim that this is the sole method of doing so, it does conclude that using SPKI/SDSI in distributed trust management for expressing simple and flexbile policies using S-expressions is useful, viable and promising.

As Professor Denise Anthony, of the Department of Sociology at Dartmouth observed,

> *"SPADE is able to capture the necessary features of most of modern organization (hierarchical structure) and the benefits of hierarchy, without also capturing the costs/negative aspects of hierarchy. This is very cool."*

6 Future Work

I have implemented a prototype of SPADE. Consequently, there is a lot of scope for future work in SPADE working with Shibboleth. Some of the possible things are as follows:

1. For the prototype, I have not implemented Certificate Revocation Lists. Instead, I have assumed short certificate life spans in keeping with the true nature of SPKI/SDSI. Online revalidation may be implemented in the system.

2. SPADE allows Domain Administrators to embed 'hidden' attributes in their ARP certificates which the Domain user is not aware of. This can be changed to inform the user of the nature and / or purpose of those hidden attributes. Moreover, information on the user's site may be made available to the user detailing what type of ARPs are required by specific sites to be able to release resources to users.

3. Also a time and efficiency analysis of certificate retrieval, intersection and policy resolution must be done. These may be done using different load balancing in the network signifying different traffic loads and numbers of users.

4. Domain resolution, SHAR and URL resolution and Role resolution all can be done in a number of different ways. These can be implemented and compared for efficiency and effectiveness.

5. The Shibboleth standard is still being developed. Future developers can look at authentication means using SPKI/SDSI certificates from this infrastructure for authentication with public keys to connect to Shibboleth.

6. Since SPADE is a SPKI/SDSI based Privilege Management Infrastructure, it is important that a means of connecting an authorization system like this with a standard PKI system is researched. Since X.509 is the popular system used today, future re-

search can delve on how to connect a SPKI/SDSI PMI system with a X.509 PKI system.

A SPADE code details

SPADE uses Java, HTML, and Java Servlets. The code is divided into three main parts:

1. SPKI/SDSI code. This is part of Java code developed by the Cryptography and Information Security Group at the Massachusetts Insitute of Technology [MIT]. I have used the 'sdsi' java package developed by Alexander Morcos and Sameer Ajmani. It is accesible at [MIT]. 'sdsi' is a GUI package designed to familiarize users with SDSI operations such as creating certificates and deriving authorization decisions from certificate chains. I have made a large number of modifications to the code. I have written approximately **1250 lines of Java code** to add to this package.Some of them are as follows:

   (a) Stripping away the GUI code and adding helper functions in relevant classes.

   (b) Implementing a tag intersection library: The 'sdsi' Java package did not have a library to intersect SPKI/SDSI certificate tags. I have written a tag intersection library that is based on the SPKI/SDSI standard [Ellison2] and processes normal as well as special s-expressions such as the wildcarded (*), (* range), (* prefix) and (* set) formats.

   (c) Writing helper files that are used to process SPADE relevant operations quickly such as obtaining the names of all roles in a Domain by resolving it's Authorization and Name Certificates, etc.

2. Java Servlet code: This is the bulk of the SPADE code. It plugs into the SPKI/SDSI code to create objects such as public keys, Name certificates, and Authorization certificates and to verify certificate chains. It is used to display the web based GUI constituting dynamic web pages, to process user input and to interact with the prototype databases (which are actually file system directory hierarchies). I have written approximately **4000 lines of Java Servlet code**.

3. HTML code: I have also written a few pages of HTML code to be able to call and process servlets. I have written approximately **300 lines of HTML code**.

The entire code package and directory hierarchy (approximately 5500 lines of code) sits on a server (http://zermatt.dartmouth.edu), where it is plugged into a Shibboleth test club implementation (Shibboleth v 0.8), implemented by Omen Wild at the Dartmoutn PKI lab.

B Biographies

B.1 Sean Smith (Advisor and Committee Chair)

Taken from [Smith] :

"Assistant Professor

Department of Computer Science, Dartmouth College

Prof. Sean Smith has been working in information security—attacks and defenses, for industry and government—for over a decade. In graduate school, he worked with the US Postal Inspection Service on postal meter fraud; as a post-doc and staff member at Los Alamos National Laboratory, he performed security reviews, designs, analyses, and briefings for a wide variety of public-sector clients; at IBM T.J. Watson Research Center, he designed the security architecture for (and helped code and test) the IBM 4758 secure coprocessor, and then led the formal modeling and verification work that earned it the world's first FIPS 140-1 Level 4 security validation. Dr. Smith has published numerous refereed papers; given numerous invited talks; and been granted nine patents. His security architecture is used in thousands of financial, e-commerce, and rights managements installations world-wide.

In July 2000, Sean left IBM for Dartmouth, since he was convinced that the academic education and research environment is a better venue for changing the world. His current work, as PI of the Dartmouth PKI Lab, investigates how to build trustable systems in the real world.

Dr. Smith was educated at Princeton and CMU, and is a member of ACM, USENIX, the IEEE Computer Society, Phi Beta Kappa, and Sigma Xi."

B.2 Edward Feustel

Taken from [ISTS]:

"Ed Feustel became a Research Associate at Dartmouth's Institute for Security Technology Studies in August 2000. Ed's research focuses on security issues in distributed computing applications and infrastructure. He has represented Prime Computer and the Institute for Defense Analyses as technical liaison to the Object Management Group, Unix International, and The Open Group (OSF and X/Open)

He comes to Dartmouth from the Computer and Software Engineering Division of the Institute for Defense Analyses, Alexandria, Virginia where he was a Research Staff Member with interests in Security and Distributed Systems and Applications. Prior to IDA Alexandria, he worked at Prime Computer as a Principal Technical Consultant, Rice University as a tenured Associate Professor of Electrical Engineering and Computer Science, IDA Princeton as a Systems Programmer, Lawrence Livermore Laboratory as a Sabbatical Researcher, and California Institute of Technology as a Research Fellow. He is a graduate of Princeton University (MA and Ph.D. in Electrical Engineering) and MIT (BSEE and MSEE). His 1971 IEEE Transactions of Computers Paper on the Advantages of Tagged Architectures coined the term "self identifying data" using tags or descriptors and set the stage for network/application exchange of tagged data by protocols used in OMG's CORBA and XML."

B.3 John Erickson

Taken from [Erickson]:

" John S. Erickson is a Principal Scientist with Hewlett-Packard Laboratories, where current research touches on may areas related to policy expression, enforcement and compliance for distributed systems, including open architectures for digital rights management and privacy protection; he also has a long-standing interest in digital object repository architectures. John was awarded a U.S. patent in 1998 for rights management technologies and services that originated with his Ph.D. research at Dartmouth College; other related patents are pending. He has been an active participant in various international metadata and rights management standardization efforts, and sits on a number of working groups and advisory panels. John holds a Ph.D. in Engineering Sciences from Dartmouth College (1997), an M.Eng.(EE) from Cornell University (1989), and a BSEE from Rensselaer Polytechnic Institute (1984). He founded NetRights LLC in 1995 and was VP-Technology Strategy for Yankee Rights Management, 1997-1999."

References

[Ajmani] "Sameer Ajmani's Home Page. SDSI projects". Massachusetts Institute of Technology. <http://pmg.lcs.mit.edu/~ajmani/sdsi>

[Ashley] P. Ashley et al. "Enterprise Privacy Authorization Language". <http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/index.html>

[Blaze1] M. Blaze, J. Feigenbaum, and J. Lacy. "Decentralized Trust Management". In Proceedings of the 1996 IEEE Symposium on Security and Privacy, pages 164-173. November 1996. Los Alamitos.

[Blaze2] M. Blaze, "The Role of Trust Management in Distributed Security". In Secure Internet Programming: Issues in Distributed and Mobile Object Systems, volume 1603, pages 185-210.

[Blaze3] M. Blaze, J. Feigenbaum, and M. Strauss. "Compliance Checking in the PolicyMaker Trust Management System". In Proceedings of the Financial Cryptography Conference, 1998.

[Blaze4] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. "The KeyNote Trust Management System - Version 2". Internet Engineering Task Force, September 1999. RFC 2704

[Chadwick1] D. Chadwick. "The PERMIS X.509 Based Privilege Management Infrastructure." Internet Draft: <draft-irtf-aaaarch-permis-00.txt>,

[Chadwick2] D. Chadwick. "The PERMIS X.509 Role Based Privilege Management Infrastructure". 7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002).

[Chadwick3] E-mail correspondence with David Chadwick

[Clarke] D. Clark, J. Elien, C. Ellison, M. Fredette, A. Morcos, and R. Rivest. "Certificate Chain Discovery in SPKI/SDSI". Journal of Computer Security, 9(4): 285-322, 2001.

[Cranor] Lorrie Cranor, Marc Langheinrich,Massimo Marchiori,Martin Presler, Joseph Reagle. *"The Platform for Privacy Preferences 1.0 (P3P1.0) Specification"*. W3C Recommendation. 16 April, 2002 <http://www.w3.org/TR/2002/REC-P3P-20020416/>

[Dulay] N. Dulay, E. Lupu, M. Sloman, and N. Damianou. "A Policy Deployment Model for the Ponder Language". In Proceedings of 7th IFIP/IEEE International Symposium on Integrated Network Management : Integrated Network

Strategies for the New Millennium. 14-18 May 2001. Seattle, Washington, USA.

[Eastlake] D. EastLake 3rd, J. Reagle, D. Solo. *“(Extensible Markup Language) XML-Signature Syntax and Processing, RFC 3275”* <http://www.ietf.org/rfc/rfc3275.txt>

[Ellison1] Carl M. Ellison, *“Naming and Certificates”*. Proceedings of the 10th Conference on Computers, Freedom & Privacy. “Computers, Freedom & Privacy 2000: Challenging the Assumptions”. Westin Harbor Castle, Toronto, Ontario, Canada, April 4- 7, 2000.

[Ellison2] Carl M. Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brain Thomas and Tatu Ylnen. *“SPKI Certificate Theory”*, RFC 2693, IETF Network working group, September 1999. <http://www.ietf.org/rfc/rfc2693.txt>

[Ellison3] E-mail correspondence with Carl Ellison

[Erickson] E-mail correspondence with John Erickson

[Godik] Simon Godik et al. ”*OASIS eXtensible Access Control Markup Language (XACML)*”. OASIS Standard, 18 February 2003. <<http://www.oasis-open.org/committees/download.php/1642/oasis-####-xacml-1.0.pdf>>

[Hedberg1] R. Hedberg and T. Wiberg. "Why SPOCP?", SPOCP Project document, , <ftp://ftp.su.se/pub/spocp>

[Hedberg2] R. Hedberg and T. Wiberg. "The SPOCP protocol", SPOCP Project document. <ftp://ftp.su.se/pub/spocp>

[Hedberg3] R. Hedberg and T. Wiberg. "Creating Rules", SPOCP Project document <ftp://ftp.su.se/pub/spocp>

[Hedberg4] R. Hedberg and T. Wiberg. "SPOCP: Use of S-expressions", SPOCP Project document. <ftp://ftp.su.se/pub/spocp>

[Hensley1] Pat Hensley,Max Metral, Upendra Shardanand, Donna Converse, Mike Myers. *“Implementation of OPS Over HTTP”* Version 1.0, June 2, 1997. <http://www.w3.org/TR/NOTE-OPS-OverHTTP.html>

[Hensley2] Pat Hensley,Max Metral, Upendra Shardanand, Donna Converse, Mike Myers. *“Proposal for an Open Profiling Standard”* Version 1.0, June 2, 1997. <<http://www.w3.org/TR/NOTE-OPS-FrameWork.html>>

[Herzberg] A. Herzberg, Y. Maas, J. Mihaeli, D. Naor, and Y. Ravid. “Access Control Meets Public Key Infrastructure, Or: Assigning Roles to Strangers”. In Proceedings of the 2000 IEEE Symposium on Security and Privacy, pp. 2-14, 2000. IEEE Press.

[ISTS] Institute for Security Technology Studies. <http://www.ists.dartmouth.edu/cybersecurity/whoweare.htm#feustel>

[Kagal] L. Kagal, T. Finin, and Y. Peng. "A Framework for Distributed Trust Management". In Proceedings on IJCAI - 01 Workshop on Autonomy, Delegation and Control, 2001.

[Kaufman] Charlie Kaufman, Radia Perlman, Mike Speciner, *“Network Security: Private Communication in a Public World”* , Englewood Cliffs, NJ Prentice Hall, 1995. ISBN: 0-13-046019-2

[Madsen1] Paul Madsen, Carlisle Adams.*“Privacy and XML, Part 1”*. <http://www.xml.com/lpt/a/2002/04/17/privacy.html>

[Madsen2] Paul Madsen, Carlisle Adams.*“Privacy and XML, Part 2”*.<http://www.xml.com/lpt/a/2002/05/01/privacy.html>

[MIT] “Cryptography and Information Security Group Research Project: SDSI”. Massachusetts Institute of Technology. <http://theory.lcs.mit.edu/~cis/sdsi.html>

[Morcos] A. Morcos. “A Java Implementation of Simple Distributed Security Infrastructure”. Master’s Thesis. Massachusetts Institute of Technology, May 1998.

[Nazareth] S. Nazareth. “Implementing authorization mechanisms in Shibboleth using SPKI/SDSI”. Master’s Thesis Proposal. Department of Computer Science, Dartmouth College, November 14th, 2002.

[Nykanen] Toni Nykanen, “Attribute Certificates in X.509”, Department of Computer Science, Helsinki University of Technology <http://www.hut.fi/~tpnykane/netsec/final/>

[Maler] Eve Maler et al. *“SAML v1.0, Assertions and Protocol*”. < <http://www.oasis-open.org/committees/download.php/1371/oasis-sstc-saml-core-1.0.pdf> >

[Rivest1] Ronald L. Rivest, Butler Lampson, *“SDSI - A Simple Distributed Security Infrastructure”,* (MIT Laboratory for Computer Science and Microsoft Corporation, April 1996) <http://theory.lcs.mit.edu/~rivest/sdsi10.html>

[Rivest2] Ron Rivest, “S-expressions” draft-rivest-sexp-00.txt, May 1997. <http://theory.lcs.mit.edu/~rivest/sexp.html>

[Sheldon] Jeffrey W Sheldon, “*Privacy Issues in Public Key Infrastructures”*. http://www.swiss.ai.mit.edu/6.805/student-papers/fall99-papers/sheldon-pki.html

[ShibDraft] M. Erdos and S. Cantor. *"Shibboleth architecture draft v05"*. <http://middleware.internet2.edu/shibboleth/docs/draft-internet2-shibboleth-arch-v05.pdf>May 2, 2002.

[Smith] Sean Smith's Home Page, <http://www.cs.dartmouth.edu/~sws>

[Vollbrecht] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, D. Spence. "AAA Authorization Framework". Internet Engineering Task Force, RFC 2904. August 2000.<http://www.ietf.org/rfc/rfc2904.txt>

[wwwP3P] *"Why Implement P3P?"* <http://www.p3ptoolbox.org/guide/section1.shtml>