XML-based Trust Management in MAS

Agostino Poggi and Michele Tomaiuolo, *University of Parma*

***Abstract*—This work deals with trust management in open and decentralized agent-based environments. Analysis and solutions are geared towards peer to peer networks, intended not just as a technology, but above all as a web of trust relationships, where parties interoperate directly, without reliance on any centralized directory or authority.**

***Index Terms*— Cooperative systems, Security, Public key cryptography, Resource management**

I. Introduction

WHILE a number of architectures and systems are being proposed to deal with the problem of service composition, some issues remain open. In particular, multi-agent systems allow the dynamical and intelligent composition of services by means of delegation of goals and duties among partners. But these delegations can never come into effect, if they're not associated with a corresponding delegation of privileges, needed to access some resources and complete delegated tasks, or achieve desired goals.

This work deals with trust management in open and decentralized agent-based environments. Analysis and solutions are geared towards peer to peer networks, intended not just as a technology, but above all as a web of trust relationships, where parties interoperate directly, without reliance on any centralized directory or authority.

In particular, this paper will show how it is possible to join multiple XML-based certificates in a delegation chain, expressing a degree of trust between two agents. This kind of delegation can allow secure collaboration also among agents who don't have direct acquaintance.

Securing access to the resources made available by the peers is a requirement to make peer to peer networks a more widespread paradigm of cooperation among loosely coupled software agents. The secure management of trust relationships, the ability to precisely control the flow of delegated permissions to trusted entities, are a fundamental requirement to allow the composition of the more disparate services provided on the network.

II. Rethinking PKI

Traditionally, the problem of identity management was considered equivalent to PKI, and in some sense it is. However, in practice, all efforts to deploy a X.509 [1] infrastructure have all fallen below expectations. Professionals share with users a widespread bad taste about PKI. PKI is expensive and hard to manage, even harder to use for the average human, and implementations lack the broad interoperability the standards promised. [8]

According to a number of proposals (e.g. PolicyMaker [2], KeyNote [3], Simple Distributed Security Infrastructure [4], Simple Public Key Infrastructure [5]) the very foundation of digital certificates needs to be re-thought, trying to make them really useful in application scenarios. The main rationale is that what computer applications need is not to get the real-life identity of keyholders, but to make decisions about them as users. Often these decisions are about whether to grant access to a protected resource or not.

In available PKI systems, these decisions should be taken on the basis of a keyholder's name. However, a keyholder's name does not make much sense to a computer application, other than use it as an index for a database. For this purpose, the only important thing is the name being unique, and being associated with the needed information. Given this reasoning, it is extremely unlikely that the given name by which we identify people could work on the Internet, as it will not be unique.

Moreover, since the explosion of the Internet, contact with person became often only digital, without ever encountering partners personally. In this cases, which are more and more common, there is no body of knowledge to associate with the name. Trying to build an on-line, global database of facts and people is obviously unfeasible, since it will face privacy problems, as well as business unwillingness to disclosure sensible data about their employees and their contacts.

A. Authorization Certificates

In fact, security cannot be founded just on identity, or given names, but more appropriately on principals and authorization. In general, a principal is any entity that can be taken accountable for its own actions in the system, and in particular, principals could be simply thought as entities associated with public keys. The SPKI documentation goes even further, dealing with principals as they "are" public keys. This means that each principal must have its own public key, through which it can be identified, and each public key can be granted rights to access system resources.

A. Poggi is with University of Parma, Dipartimento di Ingegneria dell'Informazione, Viale G. Usberti 43100 Parma, Italy (poggi@ce.unipr.it).

M. Tomaiuolo is with University of Parma, Dipartimento di Ingegneria dell'Informazione, Viale G. Usberti 43100 Parma, Italy (tomamic@ce.unipr.it).

The key concept in a trust management system, in fact, is authorization, and more precisely distributed authorization. Each entity in the system has the responsibility to protect its own resources, and it is the ultimate source of trust, being able to refuse or accept any request to access the resource.

On the other end, each entity can access some resources without being listed in a comprehensive Access Control List (ACL).

In fact, relying on local authorities and delegation, ACLs can be relegated to a marginal role, while a central role is played by authorization certificates. A basic authorization certificate defines a straight mapping: *authorization -> key*.

The complete structure of a certificate can be defined as a 5-tuple:

1. *issuer*: the public key (or an hash of it) representing the principal who signs the certificate;
2. *subject*: the public key (or, again, an hash, or a named key) representing the principal for whom the delegation is intended to; other types of subjects are allowed, but they can be always resolved to a public key; for example, a threshold subject can be used to indicate that k of n certificate chains must be resolved to a single subject (i.e. to a public key) to make the authorization valid;
3. *delegation*: a flag to allow or block further delegations;
4. *authorization*: an s-expression which is used to represent the actual permissions granted by the issuer to the subject through the certificate;
5. *validity*: the time window during which the certificate is valid and the delegation holds.

Thus, through an authorization certificate, a manager of some resources can delegate a set of access rights to a trusted entity. This newly empowered principal can, on its side, issue other certificates, granting a subset of its access rights to other entities. When finally requesting access to a resource, the whole certificate chain must be presented. Precise algorithms are presented in the SPKI proposal to combine certificates in a chain and to solve them to an authorization decision.

It can be easily noted that, in the whole process of delegation, identities and given names never appear. Keyholder names are certainly important, and careful identification is obviously a necessary condition before delegation can be granted. Otherwise principals (i.e. public keys) cannot be associated with the humans ultimately responsible for their actions.

But the interesting thing is that this association is never used in the authorization process, as in fact it is not necessary. The result is a radical simplification of the whole security infrastructure. Also, the whole system is much more flexible, allowing arbitrary delegation of permissions and anonymous communications (in the sense that user's identity is never communicated through the network). Above all, trust chains are made part of the system, being its real core, and they can be easily traced following the chains of authorization certificates issued by the involved principals.

B. Name Certificates

The Simple Digital Security Infrastructure (SDSI) [4], which eventually became part of the SPKI [5] proposal, showed that local names could not only be used on a local scale, but also in a global, Internet-wide, environment. In fact local names, defined by a principal, can be guaranteed to be unique and valid in its namespace, only. However local names can be made global, if they are prefixed with the public key (i.e. the principal) defining them.

A convention of SDSI is to give names defined in a certificate a default namespace, being the issuer of the certificate itself. Otherwise local names have always to be prefixed with a public key which disambiguates them. When used in this way, names become Fully Qualified SDSI Names. Compound names can be built by joining local names in a sequence. So, for example, PK1's Joe's Bill can be resolved to the principal named Bill by the principal named Joe by the principal (holding the public key) PK1.

Another type of SPKI certificates is defined for associating names with their intended meaning: *name -> subject*. A SPKI Name Certificate doesn't carry an authorization field but it carries a name. It is 4-tuple:

1. *issuer*: the public key (or an hash of it) representing the principal who signs the certificate;
2. *name*: a byte string;
3. *subject*: the intended meaning of the name; it can be a public key or another name;
4. *validity*: the time window during which the certificate is valid and the delegation holds.

There's no limitation to the number of keys which can be made valid meanings for a name. So in the end, a SPKI name certificate defines a named groups of principals. Some authors [9] interpret these named groups of principals as distributed roles.

C. SAML Overview

The main scope of this work is an integrated environment, where multi-agent systems, as well as systems built on different models and technologies, can interoperate both providing and accessing services. For this purpose, it is also important to use security models which can enable a corresponding interoperability with regard to the management and delegation of privileges, allowing trusted partners to access protected resources even when their particular application is founded on different models and technologies.

The Security Assertion Markup Language [6], being standardized by OASIS, is an open, XML-based format to convey security information associated with a principal. While SAML allows to exploit digital signature and PKI technologies, its specifications are not about the deployment of some PKI, but about their use in a federated environment along with other technologies. The Liberty Alliance, for example, concentrates its work on SSO, to allow the use of services from different providers without repeating the login operations at every site.

The approach of SAML is radically different from X.509, above all as its specifications start from realistic use cases, which deal with problems that traditional, X.509 based, PKI was never able to solve. The lack of attention to real world cases is probably one of the worst characteristics of X.509. SAML and federated identity, instead, deal with the problem of system security following a bottom-up software engineering approach, taking into account already existing infrastructures.

Instead of defining, and imposing, a top down model, SAML and federated security credentials enable already deployed systems to grow and join others, on the basis of precise and limited agreements. This way, the experience gained in the implementation and deployment of security infrastructures is not lost. On the contrary, it is the basis for the new generation of integrated security systems.

Moreover, SAML is based on XML, and so it easily integrates with Web-services and other XML based applications. It can leverage existing standards and protocols, like XML Digital Signature, XML Encryption, SOAP, WSDL and WS-Security.

SAML itself deals with three different kinds of assertions:

- authentication assertions;
- attribute assertions;
- authorization decision assertions.

Authorization decision assertions are a somehow "frozen" feature in current specifications, suggesting a better solution is to rely on other available standards for security policies, like XACML. A profile to integrate XACML authorization decisions into a SAML assertion has been standardized with SAML 2.0.

The three types of assertions are issued, in principle, by three different authorities. The Policy Enforcement Point (PEP) represent the component of the system which takes care of analyzing provided assertions, and generate authorization decisions, about whether to grant or to deny access to a protected resource.

The generic structure of a SAML assertion makes evident it is very similar to what is usually called a "digital certificate". Like in every other certificate, an issuer attests some properties about a subject, digitally signing the document to prove its authenticity and to avoid tampering. Conditions can be added to limit the validity of the certificate. As usual, a time window can be defined. Moreover, it can e limited to a particular audience or to a one-time use. Conditions can also be put on the use of the certificate by proxies who want to sign more assertions on its basis.

D. *SAML from the "Trust Management" Perspective*

Being designed to allow interoperability among very different security systems, SAML offers a variety of schemes to format security assertions. In particular, there are a number of possible ways to represent a subject, which also allow to keep away X.500 directories and DN names.

One interesting possibility is to use a SubjectConfirmation object to represent a subject directly by its public key, which resembles the basic concepts of SPKI, where, at the end, principals "are" always public keys.

Thinking about the use of SAML as a representation of SPKI authorization certificates, it would be important to have access rights, or permissions, associated with the subject. Simple authorization decisions could be encoded directly in SAML assertions till version 1.1. In the latest specifications, these assertions are considered "frozen", even if not yet deprecated. However, the very same specifications suggest alternative schemes, first of all integrating an XACML policy into a SAML assertion. The precise way to accomplish this is described in a separate profile, which will be briefly discussed in the following pages.

But, apart from direct delegation of permissions, SPKI-like trust management frameworks can also be used to implement distributed RBAC access control systems, as discussed in [LI2]. For this purpose, local names are particularly important, as they allow each principal to manage its own name space, which, on the other hand, is also one of the foundations of "federated identity" and SAML.

In fact, while SAML allows the use of X.509 distinguished names, it also support a number of other heterogeneous naming schemes. In this sense, its reliance on XML for assertion encoding is not irrelevant, as it provide intrinsic extendibility through schemas and namespaces.

Assigning a local name to a public key, or to a set of public keys, is as simple as defining a role, as in SAML names, and roles, are not considered globally unique by design. And also assigning a named principal to a local name, or to a role, is perfectly possible.

E. *XACML Overview*

The eXtensible Access Control Markup Language (XACML) [7] is a language for specifying role or attribute based access control policies. It is standardized by the OASIS group and, at the time of this writing, its latest release is 2.0.

A high level model of the XACML language is shown in the following picture. Its main components are:

- *Rule* – the basic element of each policies;
- *Policy* – A set of rules, together with the algorithms to combine them, the intended target and some conditions;
- *Policy set* – A set of policies, together with the algorithms to combine them, the intended target and some obligations.

In particular, each XACML rule is specified through its:

- *target* – indicating the resources, the subjects, the actions and the environment to which the rule applies;
- *effect* – can be Allow or Deny;
- *condition* – can further refine the applicability of the rule.

F. *XACML from the "Trust Management" Perspective*

As described in the previous sections, trust management is based on public keys as a mean to identify principals, and on authorization certificates to allow delegation of access rights among principals. SAML defines some rudimentary structures to convey authorization decisions in assertions. However, these structure are not able convey all the information that can be represented using the XACML language. On the other hand, XACML lacks means to protect requests and responses of its Policy Enforcement Points (PEP). It is clear, and so it appeared to both the SAML and the XACML working groups, that the two languages were in many senses complementary, and thus a SAML profile of XACML was defined. It effectively makes the two languages work together in a seamless way.

From the "trust management" perspective, the conjunction of SAML and XACML, in particular the inclusion of XACML authorization decisions into SAML assertions, provides a rich environment for the delegation of access rights. From this point of view, the fact that logic foundations of the XACML language exist is very important, as they provide XACML with a clear semantic. The problem is to find algorithms through which the combination of permissions granted in a chain of certificates could be computed in a deterministic way, as it is already possible in SPKI.

In fact, even if the semantic of a XACML policy is logically sound, nevertheless subtle problems can appear when different policies are linked in a chain of delegation assertions. One major problem is about monotonicity of authorization assertions, which cannot be guaranteed in the general case.

Using XACML authorization decisions as SAML assertions, it is possible to assert that access to a particular resource is denied, instead of allowed. Though being a perfectly legal and meaningful concept, the denial of a permission (a "negative permission") is is not desirable in decentralized environments. In this case, a service provider can never allow access, as it cannot be sure to possess all issued statements. On the other hand, the non-monotonicity of the system can also lead to attacks, as issued assertions can be prevented to reach the provider, this way leading it to take wrong authorization decisions.

Therefore, it is necessary to define a specific profile of SAML and XACML which could enable the secure delegation of permissions in decentralized environments. One of the first requirements is to make "negative permissions" illegal.

III. Implementation

The first step to implement the architecture described in the previous sections consisted in evaluating available software tools which can manipulate SAML and XACML structures. Unfortunately, probably due to the relative of relevant standards (especially for their latest versions), the software park is not particularly vast.

With regards to SAML, the choice fall on the OpenSAML library. In fact, while still being in a development phase, it is the only one supporting all functionalities of SAML 2.0 and, above all, allowing to define new classes with relative simplicity. Extensibility is in fact particularly important, in our case, to realize a "glue" level between SAML and XACML, embodied by the *XACMLPolicyStatement* element.

About XACML, instead, the choice of Sun's XACML Implementation was obliged, in practice, as it's the only valid open source tool to deal with the language. Anyway, a little explaining is necessary. In fact, to handle the version 2.0 of XACML, the CVS version of the library must be used. Such version, anyway, presents a small defect which, at this moment, has not been corrected yet. In fact its APIs miss a method to obtain the list of targets of policies. The author of the library, contacted about this, confirmed the missing and suggested us to implement the method (which we did) waiting for the definitive release of the 2.0 version of the library.

Then, it was decided to give a standard structure to our library, realizing its API like a Java security provider. The Java Cryptographic Architecture (JCA) foresees in fact the possibility to realize packages, called security provider, which provide JDK with a concrete implementation of a subset of Java cryptograohic functionalities. For developers wanting to use the library, the main advantage of this choice is the availability of a set of API with a well known and collaudated structure. Moreover, this will allow the use of certificates and paths which will be realized with normal Java API, without duplicating their functionalities. In fact, in principle any component (also external ones), operating on a Java certificate, will be able to operate on SPKI certificate of the new library, too.

To realize an extension of the Cryptographic Architecture (JCE), first of all it was necessary to extend Java basic data types, which in our case are represented by certificates and paths; then engine classes had to be realized, which specify algorithms to be implemented. Finally, a master class for the provider had to be implemented, which is necessary to register new classes and allow them to be used by Java.

A. Certificates

To represent certificates, Java cryptographic APIs define an abstact class: *Certificate*. Within it, all basic methods to manage public key certificates can be found.. Extending this class, an abstract class, *SPKICertificate*, has been realized, containing common methods of name certificates and authorization certificates.

In particular, the *SPKINameCertificate* class extends *SPKICertificate* and describes a name certificate. Within it, all methods need to set the certificate subject are present, in all three possible forms: public key, namespace qualified local name and hash of a public key. Different get methods exist, corresponding to each type of subject to obtain. Moreover, the *getPublicKey()* method, defined by Java API, also returns the subect's public key.

Two methods, *getStatedName()* and *setStatedName()*, allow instead to set and read the SPKI name field, i.e. the name associated to the certificate subject by the certificate creator. In both methods just the local part of the name is needed, as the issuer's namespece is taken as a default.

```
public void sign( KeyPair keys, SignatureParameters params )
{
  // Create the SignatureBuilder and build the signature
  Signature sign = (Signature) buildSAMLObject( Signature.DEFAULT_ELEMENT_NAME );

  // Set up signature parameters
  sign.setSigningKey( keys.getPrivate() );
  sign.setSignatureAlgorithm( params.getAlgorithm() );
  sign.setCanonicalizationAlgorithm( params.getCanonicalization() );

  // Add the public key to the signature
  KeyInfo keyInfo = (KeyInfo) buildSAMLObject( KeyInfo.DEFAULT_ELEMENT_NAME );

  keyInfo.setPublicKey( keys.getPublic() );
  sign.setKeyInfo( keyInfo );

  // Link the signature and the assertion togheter
  SAMLObjectContentReference contentReference = new SAMLObjectContentReference( assertion );
  sign.getContentReferences().add( contentReference );
  assertion.setSignature( sign );

  // Delete del old marshalled assertion
  marshalledAssertion = null;
}
```

Fig. 1. A code snippet regarding certificate signature.

The *SPKIAuthorizationCertificate* class also extends *SPKICertificate*, and allows to create and manage authorization certificates.

Its most important method is *addPolicy()*, which allows to add a security policy to the certificate (in fact, according to SAML specifications, it's possible to add more security policies to a single certificate). Such policy has to be represented by a class implementing the simple *AuthorizationPolicy* interface, whose only needed functionality is to convert the policy to a Policy object, as defined by Sun's XACML Implementation. The *getXACMLPolicies()* method allows to obtain the security policies contained in the certificate, represented as Sun's XACML classes.

The *allowDelegation()* method, instead, allows to indicate an explicit delegation, as defined in the SPKI proposal. The opposite method, *denyDelegation()*, is useful only in the case you want to remove the delegation attribute from a certificate, where it's present. In fact, in the case *allowDelegation()* is not explicitly invoked, the delegation is not active.

Finally, the *getPublicKey()* method is also present, because it is required by the Certificate abstract class; however, as an explicit key is not normally present in this type of certificate, to be associated with the subject, the method always returns null.

B. Certificate Path Validation

An algorithm to evaluate the correctness of a certificate chain is described in the original SPKI proposal. To this aim, Java APIs define the *CertPathValidator* class. By means of it, through the implementation of its *validate()* method, the validity of a chain can be controlled.

Similarly to what happens for the creation of certification paths, the validation operation requires, as parameters, the *CertPath* that is meant to be verified and an implementation of the *CertPathParameters* interface, describing all parameters needed by the validation algorithm. The result of the operation will be eventually represented by an implementation of the *CertPathValidatorResult* interface.

Thus, a subclass of *CertPathValidator* had to be developed, implementing the SPKI validation algorithm. Parameters of the validation process are represented as *ValidatorParameters* objects, containing the list of keys trusted by the principal operating the verification, and possibly additional parameters.

The result of the process, in the case of SPKI certificates, is represented by a set of roles associated with the public key which terminates the chain. The information is included into a *KeyRoles* object.

The other operation to be offered by the library is that of validating a request to access a local resource. The request itself is represented by an instance of the *AuthorizationRequest* interface. Users of the library can provide different implementations of the interface, according to their needs.

It must be noted, however, that the authorization request must be formulated in compliance with the structure of security policies. Thus, the implementations of *AuthorizationPolicy* and *AuthorizationRequest* must use the same types and the same identifiers. For this reason, the

SimpleAuthorizationRequest has been implemented, as a request corresponding to security policies in use by the library.

Apart from the request, the algorithm with the list of authorization certificates to use and the list of trusted keys needed during the certificate verification process must be provided. Finally, in the case some additional conditions exist, it could be necessary to specify additional parameters for the verification process.

The validation happens through the creation of a Policy Decision Point (PDP). The Sun's XACML library provide the methods for creating such a decision block. However, to be able to obtain all needed policies, to validate the request, the PDP class of XACML uses various finder modules allowing to retrieve information. It was thus necessary to develop a finder module, called *AuthzPolicyFinderModule*, which is in charge of retrieving policies from authorization certificates provided as parameters.

During the process of creation of a PDP it is possible to insert additional finder modules. Such modules can be specified in the phase of construction of the *AuthorizationEvaluator* object and allow to extend the object's capabilities to search for information. Moreover, this way it is possible to provide the validation module with a series of local policies which are not stored within SPKI authorization certificates.

The final result of the operation is a list of *AuthorizationResponse* objects, one for each resource which was asked to be accessed. Each instance contains in its structure an identifier of the resource which it refers to, a decision value and a status code.

C. RAIS

As a first test, the library has been integrated into RAIS, a distributed system developed at University of Parma. RAIS (Remote Assistant for Information Sharing) is a peer-to-peer and multi-agent system composed of different agent platforms connected through the internet. Each agent platform acts as a "peer" of the system and is based on three agents: a personal assistant, an information finder and a directory facilitator; moreover, another agent, called personal proxy assistant, allows a user to remotely access her/his agent platform.

The RAIS system has been designed and implemented taking advantage of agent, peer-to-peer, information retrieval and security management technologies and, in particular, of three main software components: JADE, JXTA and Google Desktop Search.

Using the security library described here, a RAIS user can not only provide the permission to access his own files, but can also assign the permission to upload a new version of one or more existing files. In this case the PA informs his/her user about the updated files the first time he/she logs in. This functionality can be useful for the members of a workgroup involved in common projects or activities. Basic versioning capabilities are planned to be added to the Distribute Desktop Search system in the near future.

IV. Conclusion

Federated identities and security assertions are a novel technique to loosely couple already existing security systems, without requiring to design and deploy a whole new one, which could hardly fit the extremely heterogeneous variety of goals and requirements the different applications have.

Particular attention deserve SAML and XACML, for their wide applicability, their intrinsic extensibility, and their XML grounding, which allows them to easily fit into the existing web-based applications, as well as into new systems based on web or grid services. While being proven to have sound grounding in logical models, anyway they can be used in a distributed environment only under some restrictions. Otherwise, the combination of different assertions and policies could lead to unexpected results, or, even worse, expose the system to attacks.

References

[1] Housley, R. et al. Internet X.509 PKI Certificate and CRL Profile. IETF RFC 3280, April 2002. http://www.ietf.org/rfc/rfc3280.txt.

[2] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management". In Proc. of the 17th Symposium on Security and Privacy. 1996, pp. 164-173, IEEE Computer Society Press.

[3] M. Blaze, J. Feigenbaum, J. Ioannidis and A. Keromytis. 1999. "The KeyNote Trust-Management System Version 2." RFC 2704.

[4] Rivest, R.L., Lampson, B. SDSI - A Simple Distributed Security Infrastructure. http://people.csail.mit.edu/rivest/sdsi11.html.

[5] Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T. SPKI certificate theory. IETF RFC 2693, September 1999.

[6] OASIS Security Services (SAML) TC. http:// www.oasis-open.org/committees/security/.

[7] OASIS eXtensible Access Control Markup Language (XACML) TC. http://www.oasis-open.org/committees/xacml/.

[8] Lewis, J. "Reinventing PKI: Federated Identity and the Path to Practical Public Key Security". 1 March 2003. Available from: http://www.burtongroup.com/.

[9] Li, N. Local names in SPKI/SDSI. In Proc. 13th IEEE Computer Security Foundations Workshop, pages 2--15. IEEE Press, 2000.

[10] Li, N., Grosof, B. A practically implementable and tractable delegation logic. Proc. 2000 IEEE Symposium on Security and Privacy (Oakland CA, May 2000), 29-44.