Architectural Framework for Web Services Authorization

Sarath Indrakanti, Vijay Varadharajan, Michael Hitchens

INSS Research Group, Department of Computing
Macquarie University, Sydney, NSW 2109, Australia

Abstract. This paper proposes an authorization architecture for Web services. It describes the architectural framework, the administration and runtime aspects of our architecture and its components for secure authorization of Web services as well as the support for the management of authorization information. The paper also describes authorization algorithms required to authorize a Web service client. The architecture is currently being implemented within the .NET framework.

1 Introduction

In general, security for Web services is a broad and complex area covering a range of technologies. At present, there are several efforts underway that are striving to provide security services for Web services. A variety of existing technologies can contribute to this area such as TLS/SSL and IPSec. There are also related security functionalities such as XML Signature and XML Encryption and their natural extensions to integrate these security features into Web service technologies such as SOAP [1] and WSDL [2].

WS-Security specification [3] describes enhancements to SOAP messaging to provide message integrity, confidentiality and authentication. The WS-Trust [4] language uses the secure messaging mechanisms of WS-Security specification to define additional primitives and extensions for the issuance, exchange and validation of security tokens within different trust domains. While there is a large amount of work on general access control and more recently on distributed authorization [5], research in the area of authorization for Web services is still at an early stage. There is not yet a specification or a standard for Web services authorization. There are attempts by different research groups [6-9] to define authorization frameworks and policies for Web services. Currently most Web service based applications, having gone through the authentication process, make authorization decisions using application specific access control functions that results in the practice of frequently re-inventing the wheel. This motivated us to have a closer look at authorization requirements for Web services and propose an authorization architecture.

In the next section, we describe our Web Services Authorization Architecture (WSAA). Section 3 discusses the benefits of the proposed architecture. We compare our architecture to related work in section 4 and then give some concluding remarks in section 5.

Indrakanti S., Varadharajan V. and Hitchens M. (2005).
Architectural Framework for Web Services Authorization.
In *Proceedings of the Joint Workshop on Web Services and Model-Driven Enterprise Information Systems*, pages 97-106
DOI: 10.5220/0002565800970106

2 Web Services Authorization Architecture (WSAA)

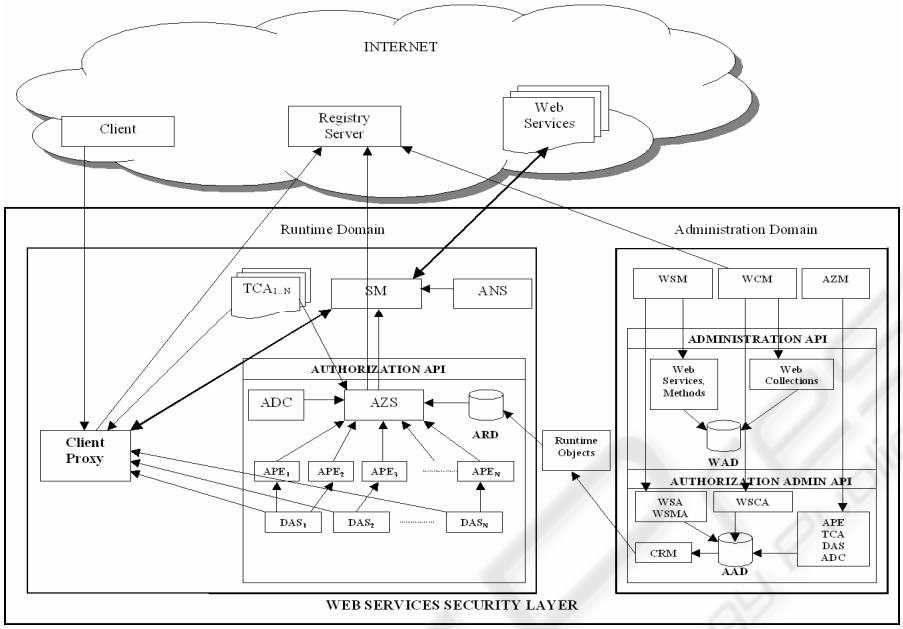

Fig. 1. Web Services Authorization Architecture (WSAA)

WSAA (figure 1) comprises of two domains - an administrative domain and a runtime domain. We manage Web services in the administration domain by arranging them into collections and the collections into a hierarchy. We provide administration support to manage a collection of Web services. We also provide support for the arrangement (adding, removing) of Web services within the collections and the movement of Web services within collections. Authorization related components such as authorization policy evaluators, trusted certification authorities (provide authentication and authorization credentials) and dynamic attribute services (provide attributes required for authorization) can be managed in the administration domain. Also security managers can assign a set of authorization policy evaluators to authorize requests to Web services.

To make the authorization process efficient, we have a runtime domain where the authorization related information such as what credentials are required to invoke a particular Web service and how to collect those credentials, is compiled and stored. This information is automatically compiled from time to time when necessary using the information from the administration domain and it can be readily used by components in the runtime domain.

The Registry Server located anywhere in the Internet is responsible for maintaining relations between services and their service providers. When a client requests the Registry Server for a specific service, the latter responds with a list of Web services that implement the requested service.

2.1 System Components

Client Proxy (CP) collects the required authentication and authorization credentials from the respective authorities on behalf of the client before sending a Web service request and handles the session on behalf of the client with a Web service's security manager.
Security Manager (SM) is an automated component responsible for both authentication and authorization of the client. A client's CP sends the necessary authentication and authorization credentials to the SM. SM is responsible for managing all the interactions with a client's CP.
Authentication Server (ANS) receives the authentication credentials from SM and uses some mechanism to authenticate the client. We treat ANS as a black box in our architecture as our focus in this paper is on authorization of the client. We included this component in the Web services security layer for completeness.
Authorization Server (AZS) decouples the authorization logic from application logic. It is responsible for locating all the authorization policy evaluators involved, sending the credentials to them and receiving the authorization decisions. Once all the decisions come back, it uses the responsible authorization decision composers to combine the authorization decisions. Where required, AZS also collects the credentials and attributes on behalf of clients from the respective trusted certification authorities and dynamic attribute services.
Authorization Policy Evaluator (APE) is responsible for making authorization decision on one or more abstract system operations. Every APE may use a different access control mechanism and a different policy language. However, an APE defines an interface for the set of input parameters it expects (such as subject identification, object information, the authorization credentials and dynamic attributes) and the output authorization result.
Trusted Certification Authority (TCA) is responsible to provide authentication and/or authorization credentials required to authenticate and/or authorize a client**.** For example, a TCA may provide public key certificates or authorization related certificates such as a Role Membership Certificate (RMC) [10].
Dynamic Attribute Service (DAS) provides system and/or network attributes such as bandwidth usage and time of the day. A dynamic attribute may also express properties of a subject that are not administered by security administrators. For example, a nurse may only access a patient's record if s/he is located within the hospital's boundary. A DAS may provide the nurse's 'location status' attribute at the time of access control. Dynamic attributes' values change more frequently than traditional static authorization credentials (also called privilege attributes). Unlike authorization credentials, dynamic attributes must be obtained at the time an access decision is required and their values may change within a session.
Authorization Decision Composer (ADC) combines the authorization decisions from authorization policy evaluators using an algorithm that resolves authorization decision conflicts and combines them into a final decision.

The *Authorization Manager* (AZM) for an organization is responsible to manage the APEs, TCAs, DASs and ADCs. S/he uses the Authorization Administration API for this purpose. The related data is stored in the Authorization Administration Database (AAD). See figure 1.

2.2 Web Services Model

We consider a Web service model based on the model defined in [7], where *Web Service*, *Web Service Method* and *Web Service Collection* are viewed as objects. Web service collections are used to group together a set of related Web service objects. Authorization related information can be managed in a convenient way if a set of related Web service objects is grouped together in a hierarchy of collections. Figure 2 shows an example of a hierarchy of Web service collections.

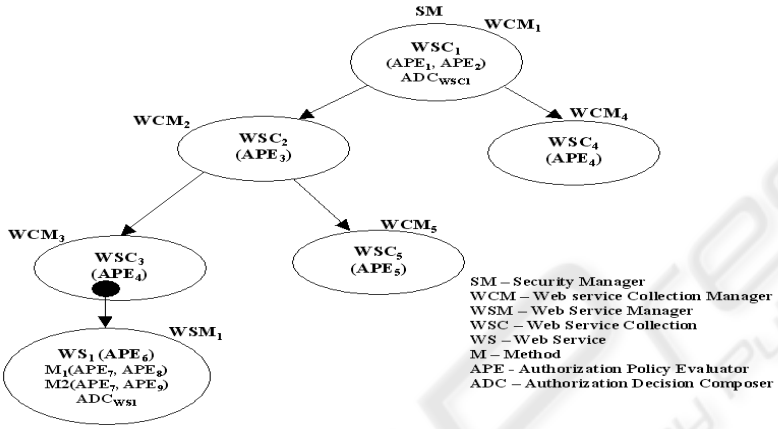

Fig. 2. Web Service Collection Hierarchy

2.3 Web Services Administration

A *Web Service Manager* (WSM) manages Web Services and Web Service Methods and a *Web service Collection Manager* (WCM) manages Web Service Collections using the Administration API (see figure 1). These objects are stored in the Web service Administration Database (WAD).

To effectively manage the collections, we arrange a set of related Web Service Collection (WSC) objects in a tree-shaped hierarchy as shown in figure 2. Each WSC in the hierarchy has a responsible Web service Collection Manager (WCM). There is only one Security Manager for a hierarchy of WSCs. In a WSC hierarchy tree, the root WSC's manager is called the *Root Web service Collection Manager* (RWCM). A RWCM is responsible for providing the Security Manager details (such as its location) in the WSDL statement of every Web service located under the collections s/he manages.

Let us consider an organization with a single hierarchy (such as the one shown in figure 2) of Web service collections. In figure 2, the root WSC is WSC_1 and the RWCM is WCM_1. We can consider a newly initiated system to simply consist of the root WSC, WSC_1 and a few Web Service (WS) objects under it managed by WCM_1. WCM_1 can add new WS objects from WAD into WSC_1. S/he can delete or move WS objects within the collections s/he is responsible for. There are other issues to consider such as 1) Who decides the location of a WS object (and how is the location changed)? 2) Who decides the shape of the tree itself? There are various design

choices to consider to answer these questions. Due to space limitations, we have not included the discussion on such design choices in this paper. We will describe these design aspects in a separate paper.

2.4 Authorization Data Administration and Policy Evaluation

A Web Service Manager (WSM) is also responsible to manage the authorization related information for the Web services s/he is responsible for. We consider a Web service method to be a high-level task that is exposed to clients. Each task (method) is made up of a number of system operations. These operations can be of different abstract types. For instance, each method of a Purchase Order service may perform one or more of these three operations - Web operation, Database operation and Mail operation. Each of these operations has a responsible authorization policy evaluator. It is reasonable to assume a WSM knows the set of tasks a Web service under his/her control performs. Similarly a WSM knows the set of operations each of these tasks (methods) perform. Using the authorization policy evaluator definitions from Authorization Administration Database (AAD), WSM associates authorization policy evaluators to Web services and their methods. This association is made in the *Web Service Authorization* (WSA) and the *Web Service Method Authorization* (WSMA) objects. WSM uses the Authorization Administration API to create and manage these objects. Similarly, a *Web service Collection Manager (WCM)* manages (using Authorization Administration API) authorization policy evaluator and authorization decision composer information in a separate object called *Web Services Collection Authorization* (WSCA) for all the collections s/he manages. These objects are stored in AAD.

Similar to Web service methods, a Web service can also have one or more authorization policy evaluators responsible for the Web service itself. Web service level policies are first evaluated before its method level authorization policies are evaluated. A Web service's authorization policy evaluators evaluate Web service level authorization policies. These policies will typically not be as fine-grained as method level authorization policies. A WSM may choose to create a new authorization decision composer for one or more Web services s/he manages or may decide to use one from the set of existing authorization decision composers from AAD if it serves the purpose.

Similar to Web services and their methods, a Web service collection can also have one or more authorization policy evaluators responsible for authorizing access to the collection itself. Collection level policies are first evaluated before a Web service's authorization policies are evaluated. A Web service collection's authorization policy evaluators evaluate collection level authorization policies. These policies will typically be course-grained when compared to Web service and Web service method level policies. Every root Web service collection has an authorization decision composer associated with it responsible for combining the decisions from all authorization policy evaluators involved. The coarse-grained authorization policies for all the relevant ancestor Web service collections (of an invoked Web service) are first evaluated, followed by the Web service level authorization policies and finally the fine-grained Web service method level policies are evaluated. The course-grained policies are first evaluated before the finer-grained policies as it helps reduce the

computing cost. If the client is not authorized by a course-grained policy, access can be denied straight away. For example in figure 2, when a client invokes WS_1's method M_1, WSC_1's authorization policies are first evaluated by APE1 and APE2, followed by WSC_2 (APE_3) and then WSC_3 (APE_4) policies. If APE_1, APE_2, APE_3 and APE_4 give out a positive decision, WS_1's authorization policies are evaluated by APE_6. If APE_6 gives out a positive decision, then finally M1's authorization policies are evaluated by APE_7 and APE_8. WS_1's authorization decision composer, ADC_{WS1} combines the decisions from APE_6, APE_7 and APE_8 and if the final decision is positive, WSC_1's authorization decision composer, ADC_{WSC1} combines the decisions from APE_1, APE_2, APE_3, APE_4 and ADC_{WS1}. If the final decision from ADC_{WSC1} is positive, the client will be authorized to invoke WS_1's method M_1.

2.5 Runtime Authorization Data

We addressed who assigns (and how) authorization policy evaluators and authorization decision composers for Web services and Web service collections. The next question is, how does a client know, where necessary, how to obtain the required authorization credentials and dynamic runtime attributes before invoking a Web service? What are the responsible authorization policy evaluators (and the credentials and attributes they require), trusted certification authorities (the credentials they provide) and the dynamic attribute services (the attributes they provide)? How does the Authorization Server (AZS) know what the set of responsible authorization decision composers for a particular client request is?

To answer these questions, we have an Authorization Runtime Database (ARD) in the runtime domain. ARD consists of the runtime authorization related information required by clients and the Authorization Server. This information is exposed to clients in the form of authorization assertions defined in a *WS-Authorization Policy* statement attached to a Web service's WSDL statement. We define an XML schema for WS-Authorization Policy statement. The statement contains information about what credentials and attributes to collect and where to collect them from. However, we do not show the schema in this paper due to space limitation.

Credential Manager (CRM) is an automated component that creates and stores the authorization runtime information, in the form of objects in ARD, using the information from WAD and AAD databases. This makes the authorization process efficient as the information in ARD is streamlined for the runtime domain. CRM is invoked from time to time, when a Web service object is added or deleted to a collection, moved within a hierarchy of collections or when the shape of the tree itself changes, to update the runtime authorization information (objects) in ARD.

When a Web service object is placed and/or moved within a Web service collection in a tree, the set of authorization policy evaluators responsible for authorizing a client's requests changes. Similarly, the set of trusted certification authorities and dynamic attribute services responsible also changes. For example, in figure 2, when WS_1 moves from WSC_3 to WSC_5, the set of responsible authorization policy evaluators for WS_1's method M2 changes from {APE_1, APE_2, APE_3, **APE_4**, APE_6, APE_7, APE_9} to {APE_1, APE_2, APE_3, **APE_5**, APE_6, APE_7, APE_9}. Once the change is made, CRM is automatically invoked and it updates ARD with the necessary runtime object entries for each method of WS_1. The responsible

authorization decision composers before and after the move will still be ADC_{WSC1} and ADC_{WS1}.

2.6 Authorization Algorithms

WSAA supports three algorithms. The first, *push-model* algorithm supports authorizations where a client's Client Proxy, using WS-Authorization Policy, collects and sends the required credentials (from trusted certification authorities) and attributes (from dynamic attribute services) to a Web service's Security Manager. The second, *pull-model* algorithm supports authorizations where the Authorization Server itself collects the required credentials from trusted certification authorities and authorization policy evaluators collect the required attributes from dynamic attribute services. The third, *combination-model* supports both the push and pull models of collecting the required credentials and attributes.

An organization must deploy one of these algorithms depending on the access control mechanisms used. If all the access control mechanisms used by the set of authorization policy evaluators are based on a pull model, then the organization must deploy the pull-model algorithm. If all the access control mechanisms used are based on a push model, then the organization must deploy the push-model algorithm. However, when some of an organization's authorization policy evaluators use the pull-model and others use the push-model, the combination-model algorithm must be deployed.

3 Discussion - Benefits of the Proposed Architecture

Some of the key advantages of the proposed architecture are as follows:
(a) Support for various access control models: WSAA supports different access control models including mandatory access control, discretionary access control, role-based access control, and certificate based access control models. The access policy requirements for each model can be specified using its own policy language. The policies used for authorization can be fine-grained or coarse-grained depending on the requirements. Access control mechanisms can either use the push-model or pull-model or even a combination of both for collecting client credentials.
(b) Support for legacy applications and new Web service based applications: Existing legacy application systems can still function and use their current access control mechanisms when they are exposed as Web services to enable an interoperable heterogeneous environment. Once again different access policy languages can be used to specify the access control rules for different principals. They could adopt a push or a pull model for collecting credentials. At the same time WSAA supports new Web service based applications built to leverage the benefits offered by Web services. New access control mechanisms can be implemented and used by both legacy and new Web service applications. A new access control mechanism can itself be implemented as a Web service. All WSAA requires is an end-point URL and interface for the mechanism's authorization policy evaluator.
(c) Decentralized and distributed architecture: WSAA allows a Web service to have one or more responsible authorization policy evaluators involved (each with its own

end-point defined) in making the authorization decision. The authorization policy evaluators themselves can be defined as Web services specializing in authorization. These features allow WSAA to be decentralized and distributed. Distributed authorization architecture such as ours provides many advantages such as fault tolerance and better scalability and outweighs its disadvantages such as more complexity and communication overhead.

(d) Flexibility in management and administration: Using the hierarchy approach of administering Web services and collections of Web services, authorization policies can be specified at each level making it convenient for Web service collection managers (WCM) and Web service managers (WSM) to manage these objects as well as their authorization related information. Another benefit of WSAA is that the credential manager component automatically generates runtime authorization objects.

(f) Ease of integration into platforms: Each of the entities involved both in administration and runtime domains is fairly generic and can be implemented in any middleware including the .NET platform as well as Java based platforms. The administration and runtime domain related APIs can be implemented in any of the available middleware.

(g) Enhanced security: In our architecture, every client request passes through the Web service's security manager and then gets authenticated and authorized. The security manager can be placed in a firewall zone, which enhances security of collections of Web service objects placed behind an organization's firewall. This enables organizations to protect their Web service based applications from outside traffic. A firewall could be configured to accept and send only SOAP request messages with appropriate header and body to the responsible security manager to get authenticated and authorized.

4 Related Work

Kraft proposes a model based on a "distributed access control processor" for Web services [7]. The main components in the authorization model are the gatekeeper, which intercepts SOAP requests to a Web service and one or more Access Control Processors (ACPs) that make the authorization decisions for the Web service. The gatekeeper itself can be an ACP. It also has the responsibility of authenticating the requesting client, combining the decisions from individual ACPs and to make the final access control decision. The advantage of this model is it supports decentralized and distributed architecture for access control. The model is generic enough to support different models of access control. This model however, does not provide support for administration of authorization related information. It also does not provide support to manage Web service collections and their authorization related information using standard APIs, which our architecture provides.

Yague and Troya [8] present a semantic approach for access control for Web services. The authors define a Semantic Policy Language (SPL). SPL is used to create metadata for resources (Secure Resource Representation (SRR)) and generic policies without the target resource in them. A separate specification called Policy Applicability Specification (PAS) is used to associate policies to target objects at run time dynamically when a principal makes a request. The architecture is based on the

integration of a Privilege Management Infrastructure (PMI) and the SPL language features. At run time, depending on the Source of Authorization Descriptions (SOADs) that the Source of Authorization (SOA) in the PMI is willing to provide to the client and the SRRs, the Policy Assistant component streamlines the SPL policies and the PAS. What is interesting in this model is that the authorization policies can be attached dynamically based on the metadata of the resource being accessed and also be streamlined dynamically to the SOADs the SOA is willing to send, through the PMI client. The disadvantage with this model is that authorization policies can only be written in SPL and is based on one model of access control – the PMI, which means this model is not generic enough to support different access control mechanisms required by applications in a heterogeneous environment. This means unlike our architecture, legacy applications (using their own access control mechanisms) are not supported by this model. The model also does not provide management and administration support for Web service objects.

Agarwal et al [6] define an access control model that combines DAML-S [11], an ontology specification for describing Web services and SPKI/SDSI [12], used to specify access control policies and to produce name and authorization certificates for users. Access Control Lists (ACLs) are used to specify access control policies of Web Services. Each ACL has the properties keyholder, subject, authorization, delegation and validity. Access control is defined as a pre-condition to access a Web service. When trying to access a Web service, a user sends the set of credentials needed to access the Web service. The user does this by using the ACL provided in the access control precondition of the Web service provider. The user calculates the set of certificates needed by making use of a chain discovery algorithm. If the client is authorized with the certificates provided, the Web service returns the functional outputs sought by the client. This model is a certificate based access control model and so is not generic enough to support multiple access control models. This means legacy applications exposed using Web services cannot use different models of access control they have already been using. The ACLs in this model are simple and one cannot specify fine-grained and complex authorization policies using this model. The model also does not provide management and administration support for Web service objects.

Ziebermayr and Probst discuss their authorization framework [9] for “simple Web services”. Their framework does not consider distributed authorization and assumes that Web services provide access to data or sensitive information located on one server and not distributed over the Web. The framework uses a rule based access control model where simple rules are written for components (in which Web services reside), Web services and parameters of a Web service method. A rule consists of a reference to a service definition, another reference to a user and additional rule information for parameters where necessary. When an access request comes in, the rules at these various levels are checked and an authorization decision is made. This framework uses simple rule based access control and so does not support different models of access control. This means legacy applications cannot be exposed as Web services. Another disadvantage with this framework is that it cannot support authorizations for distributed Web services, which have access to data and/or information over a number of Web servers. Unlike our architecture, there is no abstraction of each Web service method’s function into a set of operations. This abstraction makes it easy to perform authorization administration as discussed earlier.

5 Concluding Remarks

We proposed an authorization architecture for Web services - WSAA. We described the architectural framework, the administration and runtime aspects of our architecture and its components for secure authorization of Web services as well as the support for the management of authorization information. WSAA supports push-model, pull-model and combination-model authorization algorithms.

The architecture supports legacy applications exposed as Web services as well as new Web service based applications built to leverage the benefits offered by Web Services; it supports old and new access control models and mechanisms; it is decentralized and distributed and provides flexible management and administration of Web service objects and authorization information. We believe that the proposed architecture is easy to integrate into existing platforms and provides enhanced security by protecting exposed Web services from outside traffic. We are currently implementing the proposed architecture within the .NET framework.

References

1. World Wide Web Consortium (W3C), "SOAP v1.2, http://www.w3.org/TR/SOAP/," 2003.
2. World Wide Web Consortium (W3C), "Web Services Description Language (WSDL) v1.1, http://www.w3.org/TR/wsdl," 2001.
3. B. Atkinson et al, "Web Services Security (WS-Security) Specification, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/," 2002.
4. S. Anderson et al., "Web Services Trust Language (WS-Trust), http://www-106.ibm.com/developerworks/library/specification/ws-trust/," 2005.
5. V. Varadharajan, "Distributed Authorization: Principles and Practice," in *Coding Theory and Cryptology, Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore*: Singapore University Press, 2002.
6. S. Agarwal, B. Sprick, and S. Wortmann, "Credential Based Access Control for Semantic Web Services," *American Association for Artificial Intelligence*, 2004.
7. R. Kraft, "Designing a Distributed Access Control Processor for Network Services on the Web," presented at ACM Workshop on XML Security, Fairfax, VA, USA, 2002.
8. M. I. Yagüe and J. M. Troya, "A Semantic Approach for Access Control in Web Services," presented at Euroweb 2002 Conference. The Web and the GRID: from e-science to e-business, Oxford, UK, 2002.
9. T. Ziebermayr and S. Probst, "Web Service Authorization Framework," presented at International Conference on Web Services (ICWS), San Diego, CA, USA, 2004.
10. J. Bacon and K. Moody, "Toward open, secure, widely distributed services," *Communications of the ACM*, vol. 45, pp. 59-64, 2002.
11. M. B. A. Ankolekar, J. R. Hobbs,O. Lassila, D. McDermott, D. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, "DAML-S: Web Service Description for the Semantic Web," presented at 1st International Semantic Web Conference (ISWC), Sardinia, Italy, 2002.
12. C. M. Ellison, B. Frantz, B. Lampson, R. L. Rivest, B. M. Thomson, and T. Ylonen, "Simple public key certificate, http://theworld.com/~cme/html/spki.html," 1999.