In: 2005 IEEE International Conference on Web Services (ICWS 2005), Orlando, FL, July 12-15, 2005.

Extending the Security Assertion Markup Language to Support Delegation for Web Services and Grid Services

Jun Wang
C&C Research Laboratories
NEC Europe Ltd.
D-53757 Sankt Augustin Germany

David Del Vecchio
Marty Humphrey
Department of Computer Science
University of Virginia
Charlottesville, VA 22904

Abstract

Users of Web and Grid Services often must temporarily delegate some or all of their rights to a software entity to perform actions on their behalf. The problem with the typical Grid Services approach (X.509 proxy certificates) is that commercial Web Services tooling fails to recognize these certificates or process them properly. The Security Assertion Markup Language (SAML) is a standardized XML-based framework for exchanging authentication, authorization and attribute information. SAML has broadening commercial support but lacks delegation capabilities. To address this shortcoming, we exploit SAML's inherent extensibility to create a delegation framework for Web and Grid Services that supports both direct and indirect delegation. We develop a set of verification rules for delegation tokens that rely on WS-Security X.509 signatures, but do not force any trust relationship between the delegatee and the target service. We have implemented the framework on two common Web Service hosting environments: Java/Tomcat and .NET. By leveraging existing Web Services standards, we make it easier for Grid practitioners to build and consume Web and Grid Services without resorting to Grid-specific protocols.

1. Introduction*

With the convergence of Web Services and Grid computing [1][2][3], XML and Web Services standards [10][11][12][13][14] are emerging as a common approach to construct Grids [4][8][9]. In order to support the dynamic and inter-domain environment in Grids, we require a dynamic delegation [5] mechanism between entities across organizational boundaries. The delegation requirement in Grids is a kind of constrained delegation by proxy [6]. The delegatee does not receive his own privilege, but can act on behalf of the delegater with some constraints. In general, there are two categories of delegation requirements. One is direct delegation in which a grid user needs to delegate some subset of his or her privileges to another entity in one step. For example, a user who needs to manage his or her job through a grid portal may want to grant this grid portal the necessary rights to start control and remove jobs on the user's behalf. The other form of delegation is indirect delegation in which a grid user delegates a subset of his or her privileges to another entity through an agent. For example, suppose a grid user, Alice, wants to submit jobs to a super-scheduler that will schedule the jobs onto different machines on the user's behalf. Alice will first delegate her privileges to the super-scheduler and the super-scheduler will in turn delegate Alice's privileges to a local user accounts on the remote machines that actually runs jobs.

The problem with the conventional Grid approach to delegation – X.509 proxy certificates [5] – is that commercial tooling for Web Services does not necessarily recognize and properly process these certificates (typically either the form of the certificate's Distinguished Name or path validation causes problems). Even with the recent introduction of proxy certificates in the IETF, it is not clear when (or even if) this commercial support will ever occur. *The alternative approach we pursue in this work is to leverage and extend existing Web Services standards, without breaking the existing tooling, to facilitate building and consuming web and grid services across without requiring Grid-specific protocols.*

The Security Assertion Markup Language (SAML) [13] is a XML-encoded framework for exchanging authentication, subject attribute and authorization information. Unfortunately, SAML lacks delegation capabilities. Fortunately, we are able to exploit SAML's inherent extensibility to create a delegation framework for Web Services and Grid Services that supports both direct and indirect delegation. An important contribution of this work is the enumeration of the rules by which an entity can *validate* a SAML token that purports to contain a direct or indirect delegation. Our design and implementation is based on SAML 1.1, which was the most recent OASIS standard for SAML at the time of

*This work was performed while Jun Wang was a graduate student at the University of Virginia.

development (see Section 5 for a discussion of the impact of SAML 2.0). Its core specification [17] includes both a format for expressing assertions and a request/response protocol for exchanging these assertions. Most simply, an assertion is a declaration of facts about a subject made by an issuer. SAML defines three types of assertions:

- *Authentication* – the subject has been authenticated by some means at a given time
- *Attribute* – the subject is associated with the given attributes and values
- *AuthorizationDecision* – response to an access request, whether the access has been granted or denied

Our SAML delegation framework is based on the *Attribute* statement. The SAML binding specification [16] defines protocol bindings for the use of request-response messages; our implementation uses the SOAP [15] binding.

All of the key-related information in SAML conforms to the XML Signature [14] specification, which represents traditional key data like RSA as standard XML-encoded elements. For our discussion of SAML delegation, we are mainly interested in the *KeyInfo* and *Signature* elements. As their names suggest, *KeyInfo* elements can be used to express public key information while *Signature* elements document signature-related information (for example, *digestvalue*, *transform algorithm* and *signaturevalue*).

For our SAML-based dynamic delegation framework, we also leverage the Web Services Security (WS-Security) standard [11], an oft-used way to secure SOAP messages [8][9] in both Web and Grid Services. WS-Security defines profiles for several token types including Username, X.509, Kerberos and SAML [18]. We primarily describe our implementation in the Microsoft .NET environment but note that we have also implemented this framework in Java/Tomcat.

The rest of this paper is organized as follows. Section 2 shows the architecture of our SAML delegation framework, including custom assertion formats and request/response messages for direct and indirect delegations. Section 3 describes how SAML delegation assertions can be used in Web Services and details the verification procedures for direct and indirect delegation. Section 4 analyzes some of the design decisions made and technologies used in implementing our framework. Section 5 discusses several notable technologies important for this effort, Section 6 describes some related delegation research and Section 7 concludes.

2. SAML Delegation

Our SAML delegation framework primarily consists of three XML-based components: delegation assertions, protocol requests and protocol responses. All of these are derived from (and conform to) the corresponding SAML schemas [22][23]. We will see that in direct delegation, the delegatee and delegater only need to exchange a single delegation assertion, whereas for indirect delegation, the total number of delegation assertions exchanged is equal to the length of the delegation chain.

2.1 Overview

A generic scenario for SAML delegation is illustrated in Figure 1, in which a client Bob delegates his right to a Web Portal, S_1, and S_1 delegates Bob's right to another Web Service, S_2, and so on (ending with S_{n-1} delegating to S_n). Suppose Bob submits his job to S_1 and then goes offline. After that, any entity S_i ($1 \le i \le n$) can access the Web Service W and act on Bob's behalf by presenting the delegated SAML assertion chain. The delegation from Bob to S_1 constitutes *direct* delegation while any delegation from S_i to S_{i+1} ($1 \le i \le n$) is considered *indirect*.

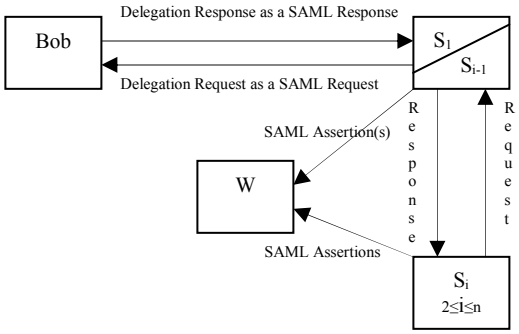

Figure 1: SAML delegation framework

2.2 Delegation Assertion

Any delegation includes 3 roles: delegater *R,* delegatee *E* and issuer *I*. For direct delegation, the issuer and the delegater are the same user. For indirect delegation, the issuer is different from the delegater. So, our delegation assertions can be expressed as: I issues an assertion declaring that R's rights are delegated to E, subject to some constraints. In Figure 2, we can see that delegation assertions consist mainly of *AttributeStatement* elements (a complete sample delegation assertion can be found at http://www.cs.virginia.edu/~jw4fr/PortalAssertion.xml). These *AttributeStatements* consist of a *Subject* element to indicate the delegatee, E, in addition to the following:

- *Issuer*: the issuer of the assertion.
- *Conditions*: *NotBefore* and *NotOnOrAfter* attributes for the lifetime of the assertion.
- *NameIdentifier*: In our implementation, this is the subject name of the delegatee's X509 Certificate.
- *ConfirmationMethod*: the verification method for establishing proof-of-possession for the subject. Two

methods are supported in SAML 1.1: *holder-of-key* and *sender-vouches* (with the WS-Security SAML Token Profile [18] supplying the recommended processing rules for each). For *holder-of-key*, the attesting entity should include an XML Signature that can be verified with the *KeyInfo* included as part of the *SubjectConfirmation* element of the assertion's subject statements. For *sender-vouches*, the attesting entity, vouches for the verification of the subject (assumes these are two different entities). The receiver must have an existing trust relationship with the attesting entity. Since we do not require a trust relationship between a delegatee and web service (Figure 1), we rely on the *holder-of-key* method.

- *KeyInfo*: the X509 certificate information for the delegatee including key name and public key info.
- *Delegation*: the identity of delegater.
- *Right*: the constraints of delegation. Set to *Full* (no constraints) if the delegater trusts the grid portal completely. Can also be set to *EndEntity,* meaning that the rights in this delegation assertion cannot be delegated further (no indirect delegation). Currently these are the only two constraints our delgation verficiation mechanisms (Section 3.2) handle, but support for more nuanced delegation constraints could certainly be added.
- *Signature*: the signature of the issuer for the assertion.

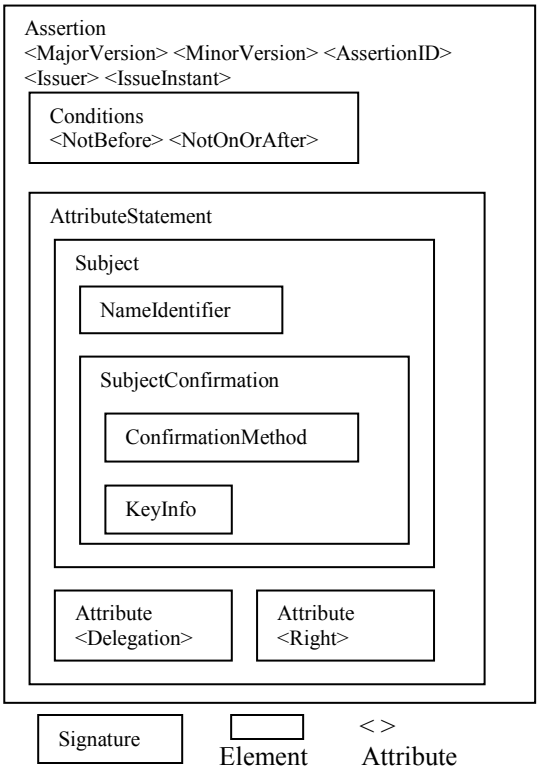

Figure 2: Delegation assertion structure

2.3 Delegation Request & Response

The delegation request and response messages conform to the SAML request and response protocol. In Figure 3, the delegatee signs a delegation request to the delegater. The delegater uses the included *Signature* element to authenticate the request. If accepted, the delegater returns a signed delegation response (Figure 4) to the delegatee (also subject to verification). The message authentication and verification procedure used by both delegater and delegatee is illustrated in Figure 5. The Assertion element included the response message (Figure 4) is a SAML delegation assertion as just described in Section 2.2.

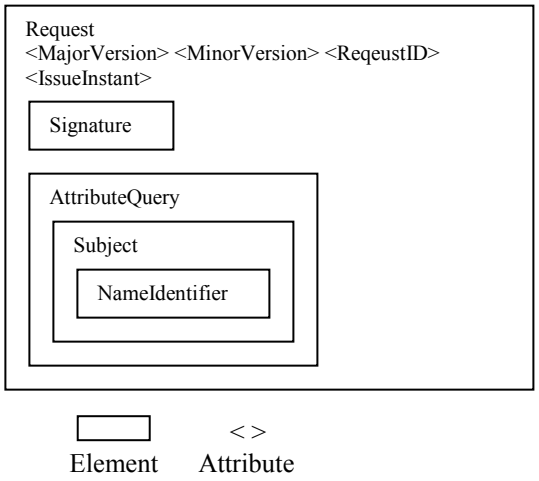

Figure 3: Delegation Request structure

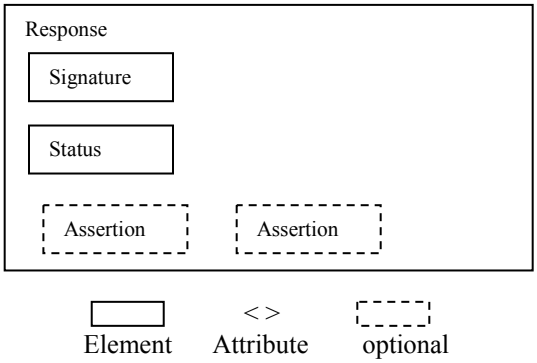

Figure 4: Delegation Response structure

In this way, the delegation assertion can be transmitted with confidence from the delegater to the delegatee. For direct delegation, the response containts a single delegation assertion; for indirect delegation, a delegation assertion is included for each delegater in the chain. Looking at Figure 1, if S_2 sends a delegation request to S_1 and wants to act on Bob's behalf, S_1's response will

contain two delegation assertions: one for S_1 issued by Bob, the other for S_2 issued by S_1. In both assertions, the *Delegation* attribute would be set to *Bob* to indicate that Bob's rights are being delegated. In Section 3, we will see that an indirect delegatee (like S_2) must present the entire assertion chain for a web service W to successfully verify that the delegate can act as Bob. Finally, note that responses to rejected delegation requests will not include any assertions, but will instead contain a *Status* element with the reason for the failed request. Complete sample request and response messages can be found at http://www.cs.virginia.edu/~jw4fr/request.xml and http://www.cs.virginia.edu/~jw4fr/response.xml.

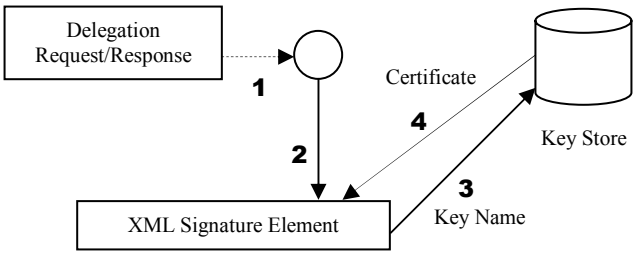

Figure 5: Request & Response authentication

3. Using SAML Delegation in Web Services

Our delegation approach is applicable to both Web Services and Grid Services. Since Grid Services can be viewed as an application and extension of standard Web Services, we are comfortable restricting our discussion in this section to the use of SAML for delegation in Web Services only. In our approach, each SAML delegation assertion is inserted as a SAML token [18] into the WS-Security SOAP header when invoking a web service method with delegation. The invoked web service method will verify the validity of the delegation using the included SAML token(s) and X509v3 signature information.

3.1 SAML Delegation with Web Service Security

The WS-Security SAML token profile [18] details how SAML assertions can be included in security headers. Since the confirmation method for delegation assertions is *holder-of-key*, a signature is needed to prove the authenticity of SAML tokens. Figure 6 (direct delegation) and Figure 7 (indirect delegation) illustrate the invocation of a web service method with delegation.

3.2 SAML Delegation Verification

One of the important challenges in effectively using SAML for delegation involves checking the validity of the expressed delegation. Our verification procedure is based the following observation: if a delegatee E wants to access a web service W on the behalf of the delegater R, then the validity of the delegation depends only on the trust relationship between W and R. Therefore, a trust relationship between W and E is not required. Below are verification processing rules for direct (Figure 6) and indirect delegation (Figure 7):

Verification Rule 1: Direct Delegation

1. The SOAP header includes a single SAML token, T.
2. The lifetime of T as expressed by the *Conditions* element must be valid.
3. To verify the delegater Bob's signature: The invoked Web Service, W extracts the key name (which is *Bob*) from the XML signature element in T. Next, W obtains Bob's public key (perhaps from a local store, W and Bob do have an established trust relationship). Finally, W will use Bob's key to verify the signature.
4. To verify S_1's valid possession of T: First, the invoked web service W extracts S_1's public key from T; then W uses this key to verify the message's X509 token profile-conformant signature.
5. No key involved in this verification can have been revoked by a Certificate Revocation List (CRL).

Only if all five conditions are satisfied can W authorize S_1 to act as Bob in a constrained way. (Specific constraints can be extracted from the assertion's *Right* attribute).

Verification Rule 2: Single Indirect Delegation

1. There are exactly two SAML tokens in the SOAP header. We call them T_1 and T_2.
2. The lifetime of each SAML token as expressed by the *Conditions* element must be valid.
3. To verify the delegater Bob's signature: First, the invoked web service W extracts the key name (which is *Bob)* from the XML signature element in T. Second, W obtains Bob's public key (perhaps from a local store), then in the third step, W verifies the signature. This matches Step 3 for direct delegation.
4. The values of *Delegation* attribute of both T_1 and T_2 must be the same.
5. The value of T_2's *Right* element must be *Full*. Our current implementation only distinguishes between two constraints: *Full* and *End Entity*. *Full* implies no constraints (i.e. the delegatee is free to further delegate the original delegater's privileges to others). *End Entity* means that only the original delegatee can act on the user's behalf; the delegatee cannot extend this right to others.
6. To verify S_1's signature in T_1: W extracts S_1's public key from token T_2, using it to check the XML signature.
7. To verify S_2's valid possession of token T_1: W extracts S_2's public key from T_1, then uses this key to verify the message signature (X.509 Token Profile).

8. All keys involved here can pass the Certificate Revocation List (CRL) check.

Rules 6 and 7 verify a delegation chain and can be easily extended to support more than 2 levels of delegation.

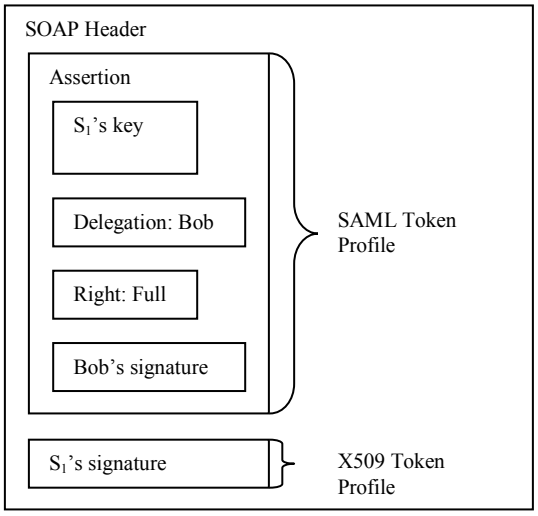

Figure 6: S_1 invokes a web service W as Bob

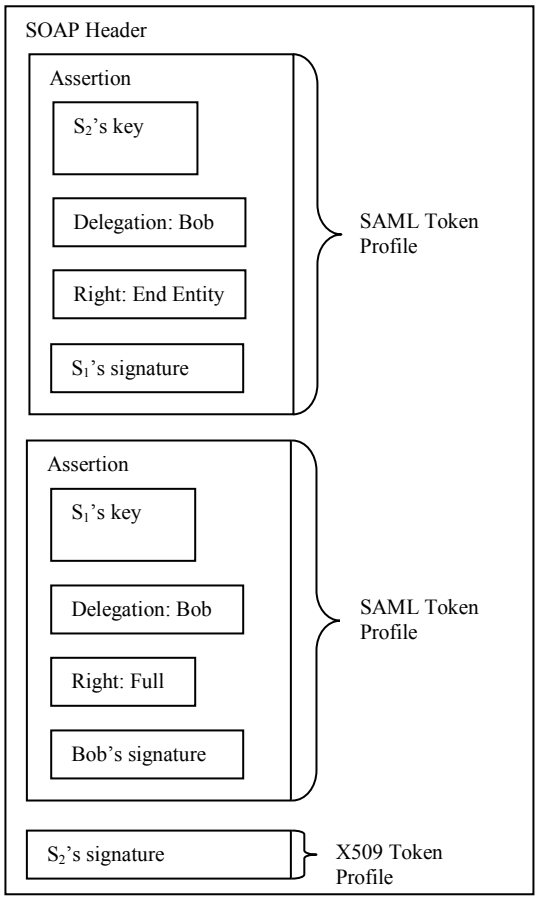

Figure 7: S_2 invokes a web service W as Bob

4. Implementation

We have implemented the SAML delegation framework on both the Microsoft .NET platform and on Java/Tomcat. Due to lack of space, we only discuss the .NET implementation here. The .NET platform includes class libraries to support XML, XML Signatures and Web Services. Microsoft also provides the Web Service Enhancements (WSE) toolkit [19] to support advanced web service features such as WS-Security, SOAP messaging and user defined XML tokens. Our implementation has components: SAMLGenerator, Delegatee, Delegater and SAMLToken. Figure 8 shows how assertions are created and consumed through protocol requests and responses between the components:

1. The Delegatee uses the SAMLGenerator to create a delegation request.
2. The Delegatee sends the request to the Delegater as a SOAP message (over HTTP or TCP).
3. The Delegater uses the SAMLGenerator to create a delegation assertion and SAML response.
4. The Delegater sends the response to the Delegatee (over HTTP or TCP).
5. The Delegatee creates SAML token(s) from the returned SAML assertions.
6. The delegatee invokes a web service method and inserts the SAML token(s) into the SOAP header.
7. The invoked web service method extracts the token as SAML token(s) from the incoming SOAP header.

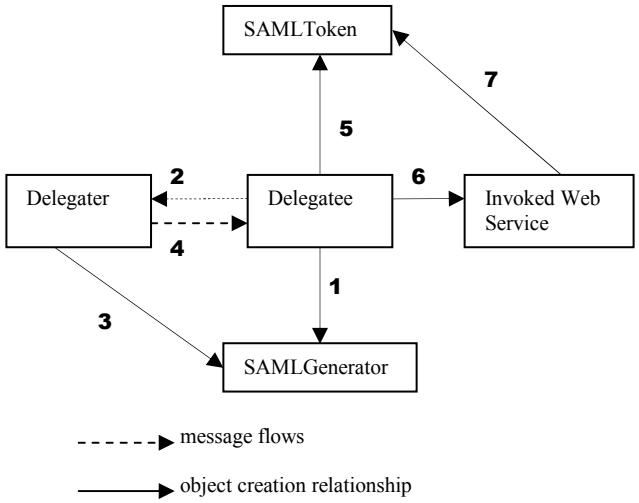

Figure 8: Component relationship

4.1 Using WSE for SAML delegation

Microsoft's WSE [19] 2.0 toolkit includes support for several evolving web services specifications including WS-Security, WS-Trust, WS-Policy and WS-Addressing.

It also supports two new features of relevance here: SOAP messaging and user defined XML token types. The SOAP messaging mechanism allows for SOAP messages to be constructed independently of the underlying transport protocol and in Section 4.2 we will show the delegatee and delegater exchanging request and response messages over TCP instead of HTTP.

WSE provides little support for WS-Security SAML tokens [18]; in fact, its support extends only to the names of elements in the SAML 1.1 assertion specification (and contains no implementations behind them). This was merely designed as an extensibility point, the idea being that users would define custom XML tokens to add the support they need. This is exactly what we have done for our SAML delegation assertions.

4.2 SAMLGenerator

The SAMLGenerator component includes classes for creating SAML Assertions, Requests and Responses. Additionally a number of utility methods are included for: verifying XML signatures, schema verification of SAML messages, delegation verification and loading/storing X.509 certificates and keys from the Windows certificate store.

4.3 Delegatee & Delegater

We developed two versions of the delegatee and delegater sender/receiver functionality. One adopts the typical web service approach (i.e. SOAP over HTTP), and the second is implemented as SOAP over TCP. The latter uses WSE's SOAP messaging mechanism (*SoapSender* and *SoapReceiver*) for message delivery. Figure 9 illustrates the message sequence between delegatee and the delegater.

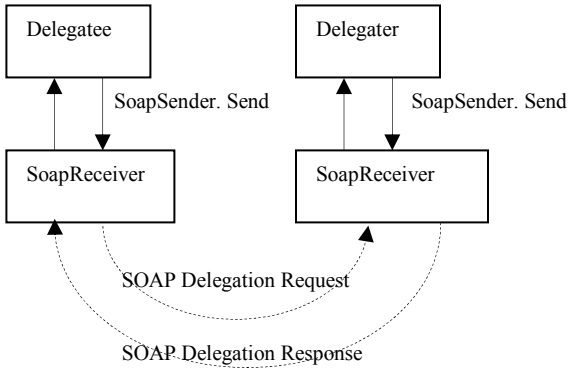

Figure 9: Communications of Delegatee and Delegater

4.4 SAML Delegation Token

As mentioned, our SAML Delegation Token Type is based on the user-defined XML token type capability of WSE. Figure 10 illustrates the processing model for user-defined XML tokens.

1. The SAML delegation token is read from the WS-Security SOAP header (SoapContext) into a run-time object.
2. The SOAP message is processed by the target web service.
3. WSE's security filter verifies every token in the SoapContext. For token types it doesn't understand (i.e. user-defined like our SAML delegation tokens) the filter will try look for custom token managers configured in the service's Web.Config file.
4. If the filter finds a matching token manager for the SAML token type, it will rely on this manager to verify the delegation assertions (the Delegation Token Manager does this according to Section 3.2).

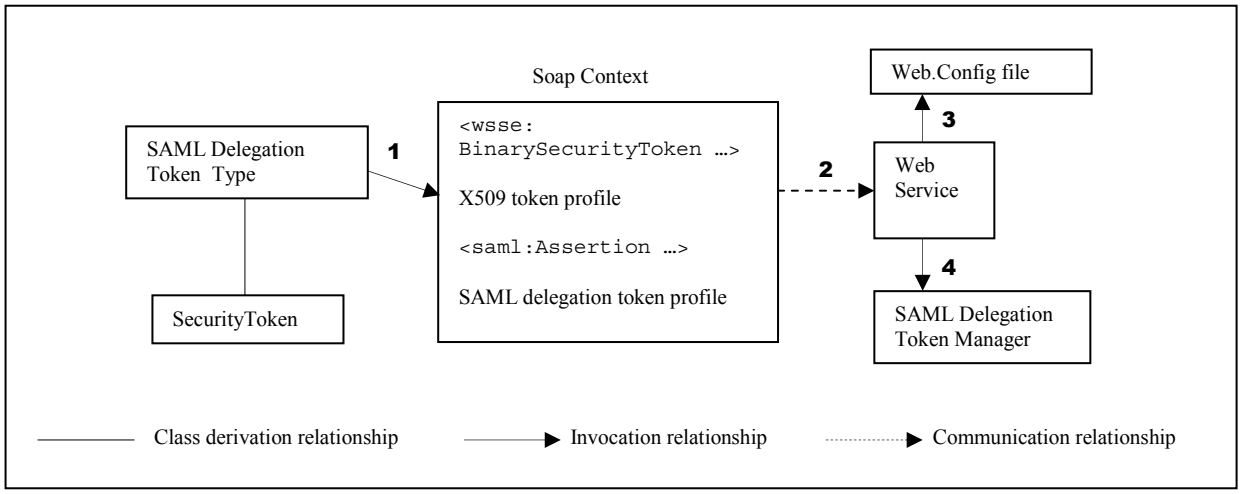

Figure 10: Processing model for user defined XML token type

A key benefit to this approach to securing Web Services is that no code changes are necessary. A change to the service's configuration file is all that is necessary for it to start processing SAML delegation tokens.

5. Discussion

There are also some other candidate standards and technologies related to our SAML delegation framework. We discuss them here briefly.

- SAML 2.0: should be stable and become a standard very soon. Compared with SAML 1.1, it adds more protocols to support some specific applications such as single sign-on. For the delegation part, the statements and attributes we use in our work do not have to change very much from SAML 1.1 to SAML 2.0. Our framework could also be (quite straightforwardly) based on SAML 2.0 once it becomes a standard.
- OpenSAML: an open source library for constructing SAML 1.1 conformant assertions and messages. It has two versions: C++ and Java. In our current implementation in Java/Tomcat, we use an HttpServlet to accept SAML requests, create assertions and send SAML responses.
- WSS4J: An open source implementation of WS-Security for Java from Apache. Recent developments include an early implementation of the SAML Token profile. On the service side, WSS4J provides the capability to process different SAML tokens type by defining Axis handlers for each type. This, combined with X509 message signature processing (X509 token profile) make WSS4J another important component of our Java implementation.

We have touched on a variety of security concerns related to our SAML delegation framework, but there are a few more worth considering. In general, our framework builds on existing security technologies (PKI-cryptography, XML signatures, WS-Security, etc.), and as a consequence, it inherits many of the strengths and weaknesses of those technologies.

Although no prior trust relationship is required between the delegatee and target service, trust between the delegater and the target service is required. We assumed that these two entities would have certificates from a common certificate authority, other ways to establish this trust do exist, but we unfortunately don't have space to discuss them here.

XML signatures are used for message integrity and WS-Security timestamp headers are assumed to help prevent replay attacks. For message confidentiality, either XML encryption or secure sockets (SSL) are the obvious choices. The latter sports better performance while the former can protect messages through intermediaries.

A final point of note is that no mechanism exists to revoke a delegation. However, the Conditions portion of delegation assertions should specify an assertion lifetime. So, we assume that delegation assertions are relatively short-lived, and are renewed or re-issued as needed. Delegation revocation and renewal are possible avenues for future research.

6. Related Work

[7] sketches a SAML assertion for constrained delegation. But the approach is different from ours. In order to support delegation, this paper extends *SubjectStatement* to *SubjectDelegationStatment* which is not supported by the SAML 1.1 or 2.0 assertion specifications. In other words, we believe that our approach is much more in line with the extensibility elements of the SAML design and can thus be supported by commercial tooling. Our approach is based on *AttributeStatement* which is supported by SAML 1.1 or 2.0 assertion specification. Our paper also presents the use of SAML delegation in Web Services and a delegation verification algorithm which combines the assertion and X509 certificate information. [7] does not discuss these issues.

Other approaches for handling delegation do exist. In V. Welch's paper [5], they define a proxy X.509 certificate format which is an extension to X.509 certificates to support delegation. This approach is currently used in the Globus project [9]. These proxy certificates are extremely valuable, but as previously mentioned, commercial tooling for Web Services does not necessarily recognize and properly process these certificates.

SAML is beginning to be deployed in other information security fields. SAML is already used in the implementation of the GT3 [9] Community Authorization Service (CAS) [21]. CAS uses *AuthorizationStatements* to represent authorization decisions. SAML is also used widely in e-commerce, especially for identity management including single sign-on. Liberty [20] is a leading identity federation and management project and makes significant use of SAML. Sun One Identity Server 6.0 [24] which implemented the Liberty protocol also supports delegation. But considering the delegation requirements of web and grid services, it makes more sense to develop an open, independent and lightweight SAML delegation solution.

7. Conclusion

Delegation is an extremely important and challenging aspect of web services security and grid services in particular. This paper presents a SAML 1.1 conformant delegation framework. Through its use in web services,

we showed the soundness of our SAML assertions in both direct and indirect delegations. We were able to build general support on both the .NET framework and in Java. With the convergence of web services and grid computing, the framework we present here can easily be integrated into any grid services built upon XML and web services standards. To our knowledge, this paper presents the first lightweight SAML-conformant delegation framework and implementation.

8. Acknowledgements

We are thankful to Von Welch of Argonne National Lab and Steven Newhouse, Deputy Director of the Open Middleware Infrastructure Institute (OMII) for helping us to clarify the presentation of several concepts in this work. The University of Virginia authors are supported in part by the US National Science Foundation under grants ACI-0203960, SCI-0438263, SCI-0426972, the Department of Energy Early Career program (to Humphrey), and the San Diego Supercomputing Center.

References

[1] I. Foster, J. Frey, S. Graham, S. Tuecke, K. Czajkowski, D. Ferguson, F. Leymann, M. Nally, T. Stoery and S. Weerawarana. Modeling Stateful Resources with Web Services. 2004. Available at http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf

[2] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke and W. Vambenepe. The WS-Resource Framework. 2004. Available at http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf

[3] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling and S. Tuecke. From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution. 2004. Available at http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf

[4] I. Foster, C. Kesselman, J. Nick and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. 2002. Available at http://www.globus.org/research/papers/ogsa.pdf

[5] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder and F. Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. In *3rd Annual PKI R&D Workshop*, 2004.

[6] O. Bandmann, M. Dam, and B. S. Firozabadi. Constrained Delegation. In *Proceedings of the IEEE Symposium on Security and Privacy,* pages 131-140, May 2002.

[7] G. Navarro, B.S. Firozabadi, E. Rissanen and J. Borrell. Constrained delegation in XML-based Access Control and Digital Rights Management Standards. 2003. Available at http://ccd.uab.es/~guille/var/ny2003.pdf

[8] WRSF.NET, http://www.ws-rf.net

[9] Globus, http://www.globus.org

[10] Web Services, http://www.w3.org/2002/ws/

[11] Web Service Security Specification. Available at http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

[12] Web Service Trust Language. Available at http://www-106.ibm.com/developerworks/library/specification/ws-trust

[13] Security Assertion Markup Language. Available at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[14] XML Signature. Available at http://www.w3.org/TR/xmldsig-core

[15] SOAP. Available at http://www.w3.org/TR/Soap/

[16] Bindings and Profiles for the OASIS SAML 1.1. 2003. Available at http://www.oasis-open.org/committees/download.php/3405/oasis-%20sstc-saml-bindings-1.1.pdf

[17] Assertions and Protocol for the OSAIS SAML 1.1. 2003. Available at http://www.oasis-open.org/committees/download.php/3406/oasis-%20sstc-saml-core-1.1.pdf

[18] Web Services Security: SAML Token Profile. 2004. Available at http://www.oasis-open.org/committees/download.php/6271/WSS-SAML-10.pdf

[19] Microsoft Web Services Enhancements, http://msdn.microsoft.com/webservices/building/wse/default.aspx

[20] Liberty Alliance Project, http://www.projectliberty.org/resources/index.php

[21] V. Welch. Use of SAML in the Community Authorization Service 2003. Available at http://www.globus.org/Security/as/Papers/SAML%20Feedback-aug19.pdf

[22] SAML 1.1 Asertion Schema. Available at http://www.oasis-open.org/committees/download.php/3406/oasis-%20sstc-saml-core-1.1.pdf

[23] SAML1.1 Protocol Schema. Available at http://www.oasis-open.org/committees/download.php/3407/oasis-%20sstc-saml-schema-protocol-1.1.xsd

[24] Sun One Identity Server, http://wwws.sun.com/oftware/products/dentity_srvr/home_identity.html

[25] OpenSAML Project, http://www.opensaml.org

[26] Apache WSS4J Project, http://ws.apache.org/ws-fx/wss4j/