

Security Assertion Markup Language Based WS-SecureConversation

Jun Wang
C&C Research Laboratories
NEC Europe Ltd.
D-53757 Sankt Augustin Germany
wang@ccrl-nec.de

Abstract

In large distributed monitoring and management systems that involve a large number of entities across multiple trust domains, the problem of establishing a secure conversation and maintaining it effectively between any two entities is outstanding when these two entities do not have a direct trust relationship. The existing Web services secure conversation work does not cover some useful scenarios involving a large number of secure conversations or delegations. In this paper, we investigate the possible mechanisms for establishing a secure conversation and describe a Security Assertion Markup Language (SAML) based secure context token format that can be used to establish secure conversations through forwarded trust relationships. By encapsulating the conversation context as well as the conversation target identity authentication information into a SAML Authentication assertion, Our SAML tokens can work effectively in these new scenarios. The underlying protocol is conformant to the emerging Web services standards of WS-Trust and WS-SecureConversation. The implementation of this framework on Java platform and its application to secure a Web services based Grid monitoring system are presented.

1. Introduction

In many large distributed computing systems like Grid, some players need to exchange several messages frequently in order to fulfill some tasks such as monitoring the system's security status, or managing the resources distributed in different locations. For instance, in a federated Web Services' Intrusion Detection System (IDS), there are many sensors and analyzers deployed in the service sites. Each sensor needs to report the security alerts it is monitoring to some analyzer through a secure communication connection. When some attacks occur, the number of exchange messages may increase greatly because the sensor may detect more alerts than in a situation without an attack (false alerts), and the analyzer may require more information from the sensor as well as the detail of some specific alerts in order to detect the attacks. Another example is the distributed Publisher-Subscriber system. A publisher may periodically publish several messages to a subscriber in a secure way. In both

examples, we can think of establishing a secure conversation connection for each communication pair.

A secure conversation (sometimes called secure context token) includes two parts: a security token and a context. The security token is used to guarantee the confidentiality and integrity of communication messages. Since a conversation usually involves multiple message exchanges between the conversation peers. A typical security token here is based on some symmetric key algorithm with a short life time. The context defines the conversation itself. The typical information includes the life time of the conversation, the conversation identity, the identities of the conversation peers, etc.

Before using a secure conversation, we need to establish it that can be trusted by the conversation peers. In terms of Web services, the emerging OASIS standards of WS-Trust [1] and WS-SecureConversation [2] are related to this topic. WS-Trust defines a Security Token Service (STS) that can issue different types of tokens to the requestors in order to establish a trust relationship between two entities. WS-SecureConversation defines a secure context token identifier that can be issued by STS and used in WS-Security [3]. It describes three establishment scenarios: security context token created by a STS, security context token created by one of the communicating parties and propagated with a message, and security context token created through negotiation/exchanges. However, it does not define the concrete establishment protocol and secure context token format. This is reasonable considering WS-SecureConversation is a generic framework to support different scenarios. In Section 5, we shall see that our SAML [4] secure context token is issued by a STS and bound to WS-SecureConversation.

SAML [4] is an XML-encoded framework for exchanging authentication, subject attribute and authorization information. Our design and implementation is based on SAML1.1, which is an OASIS standard. Its core specification [10] includes assertion, protocol request and protocol response. An assertion is defined as a declaration of claims about a subject made by an issuer. SAML provides three kinds of assertion statements: *Authentication* (the subject has been authenticated by some means at a given time), *Attribute* (the subject is associated with the given attributes and values), and *AuthorizationDecision* (response to an access request,

whether the access has been granted or denied). Our SAML context token is based on the *Authentication* statement. In this work, we exploit SAML's authentication capability to bind a secure context to the authenticated conversation peers.

The rest of this paper is organized as follows. Section 2 investigates the possible secure conversation establishment procedures and highlights some scenarios that require a new secure conversation establishment type through forwarded trust relationships. Section 3 discusses some related work on secure conversation and SAML. Section 4 defines the SAML secure context token format and shows how to use it to establish a secure conversation that can be trusted by the conversation pair. Section 5 presents the implementation of a SAML based Web services secure conversation and its application to a grid monitoring system. Section 6 discusses the possible performance improvements and expansions of our framework for different application requirements. Section 7 concludes.

2. A New Method for Secure Conversation Establishment through Forwarded Trust Relationships

We can classify the different secure conversation establishment procedures into two categories according to the trust relationship between the two conversation entities. One is called Direct Trust relationship based Secure Conversation (DTSC), in which the conversation peers have an existing direct trust relationship before establishing a secure conversation. Another is called Indirect Trust relationship based Secure Conversation (ITSC), in which the conversation peers must depend on some third parties to establish a secure conversation as well as a trust relationship. In this section, we will list the possible methods of establishment first, and then gives 2 scenarios in which the forwarded trust relationships based secure conversation can enforce.

2.1. Protocol Types for Secure Conversation Establishment

In DTSC, there are two types of establishment procedures: send-accept and negotiation as Figure 1 shows. In IDTSC, we can distinguish three different establishment types: issue-use-confirm, issue-issue-use, and issue-forward-use as Figure 2 shows. In issue-use-confirm type, a conversation requestor requests a secure context token from a trusted third party called Security Token Service (STS). The STS will issue a token for this request. The requestor will use it to protect the messages sent to the conversation target. During the first time of the conversation, the conversation target will confirm the authenticity of this conversation by requesting the same context token from its trusted STS as well according to the

context token identifier presented in each message. After that, the conversation target can verify the validity of the conversation by matching its context token with the one presented in the messages of the requestor. The requestor can verify the conversation target in the same way. Notice that if the actual trusted STSs for the requestor and the conversation target are different in IDTSC, we assume that these STSs have been federated with each other somehow using existing protocols and specifications such as Liberty Alliance Project ID-FF [8] or WS-Federation [11] so that they can be treated as one virtual STS from both the requestor and the conversation target viewpoints as Figure 2 shows.

In the issue-issue-use type, when a requestor asks for a secure context token, the STS will issue the same token to both the requestor and its conversation target directly. This establishment type requires that each conversation target must have a portal to receive any secure context token, and the portal must be registered and managed by the STS appropriately.

In the issue-forward-use type, a requestor first retrieves two secure context tokens (one for itself, another for the conversation target) from the STS. Since the tokens bind the trust relationship between the requestor and the conversation target, the requestor can establish a secure conversation and a trust relationship with the conversation target at the same time by presenting the context tokens to it. In our paper, we will present a framework that is based on this issue-forward-use type.

2.2 Scenarios for Forwarded Trust Relationships Based Secure Conversation Establishment

Our work will focus on the secure conversation for entities without direct trust relationships, and the scenarios here will be discussed in this context. Compared with the issue-use-confirm type and the issue-issue-use type, the issue-forward-use type has no direct connection between the conversation target and the STS. This is important in some scenarios.

Scenario 1: coexistence of many secure conversations
As introduced in Section 1, in the federated Web Services' IDS, or publisher-subscriber systems, the STS will issue many secure context tokens for different pairs simultaneously. For each secure conversation establishment, the issue-forward-use type needs only one connection from the STS, whereas the other types need two connections. This difference indicates not only the number of connections, but also the potential complexity of configuration, management, security, synchronization, etc.

Scenario 2: secure conversation with delegation
In the delegation case, imagine a Web Service portal: a user submits his jobs to the portal, delegates his rights to the portal, and then goes offline. The portal will run the jobs as the user in selected target machines. If the portal

needs a secure conversation with each target machine, then the issue-forward-use type is necessary, because we cannot force the delegator (the STS) to stay online before the target machine (a conversation target) can confirm the token information from it as the issue-use-confirm type does; or force the delegator (the STS) to decide an exact target machine (a conversation target) to run jobs on when the user submit his jobs to the portal (conversation requestor).

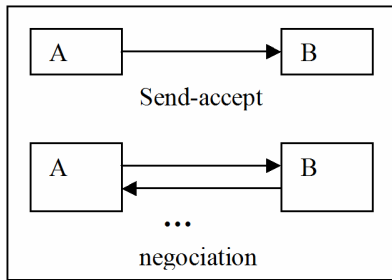


Figure 1. Establishment types in DTSC

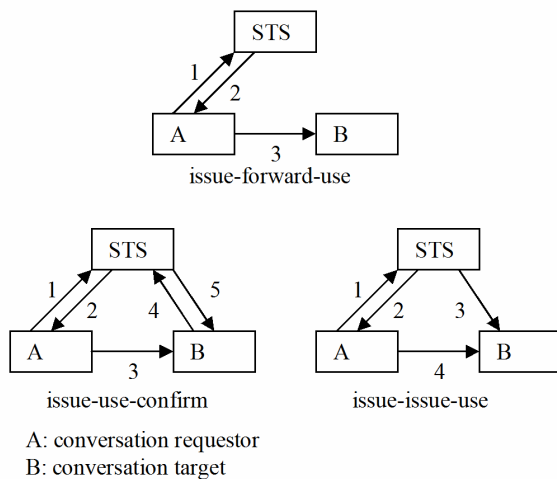


Figure 2. Establishment types in IDTSC

3. Related Work

Since WS-SecureConversation [2] is an emerging OASIS standard, most existing work is conformant to WS-SecureConversation. According to the classification of the secure conversation establishment types we introduced in Section 2, Microsoft WSE [5] supports WS-SecureConversation by issue-use-confirm type or issue-issue-use type of IDTSC, while our work is based on the issue-forward-use type of IDTSC. Another feature of our SAML context token is that since it can encapsulate the identity authentication information into the assertion as well, the conversation pair can select to renew the context token directly without the STS when the STS work load is high. WS-SecureConversation [2] provides a mechanism called Amending that is similar to the SAML assertions in that some claims can be associated with a subject using an

authenticated approach like signature. However, this mechanism is not for secure context token creation, but for context amending. Furthermore, WS-SecureConversation does not consider the secure conversation establishment scenario in which a requestor can forward the token retrieved from a STS to its peer directly. Some existing work such as Globus [6] and GEMSS [7] currently only support establishing a WS-SecureConversation conformant conversation between two entities in the same trust domain, so the two entities will create the security context token through negotiation/exchanges directly. For example, Globus [6] uses the SSL/TLS protocol for negotiation.

In this paper, the idea is to use SAML to forward trust relationships as well as establish a trust relationship for the conversation pair. SAML is used for forward or established trust relationships in other projects as well. Liberty Alliance Project [8] uses SAML to support single sign-on. A user can login and get an *Authentication* assertion from his Identity Provider and forward it to different Service Providers. Since the Service Providers already trust the Identity Provider through federation; they will trust this user as well after the user presents this assertion (without further login). SAML delegation [9] uses SAML to support delegation. A delegatee can first retrieve a SAML *Attribute* assertion from the delegator. After that, the delegatee can act on the behalf of the delegator when presenting this assertion to any Web service. In Section 6, we will discuss a derivation of our protocols that can use the idea of delegation and DTSC to support secure conversations.

4. SAML Secure Context Token Based Web Services Secure Conversation

In this section, we will investigate the capability of SAML in establishing a secure conversation for two entities using the issue-forward-use method we described in Section 2.

4.1 Overview

In Figure 2, we can see that the IDTSC establishment types include three roles: a STS and two conversation entities. In the issue-forward-use type, in order to avoid the connection of a conversation target with the STS, the STS requires the encapsulation of enough authentication information into the secure context tokens so that the conversation requestor can forward the tokens to the conversation target in an authenticable way.

Figures 3 summarize our ideas of using the SAML authentication capability to forward trust relationships as well as establish a secure conversation between two entities. First, the STS will generate a secure context token (SCT) identifier which uniquely identifies a conversation. Then it will issue two SAML authentication

assertions. The first is for the requestor, while the second is for the conversation target. Each SAML token asserts that the conversation is valid and both are authenticated by the signature of the STS. As introduced in Section 1, the basic contents that identify a conversation include a context and a security key. Both of them need to be expressed and authenticated through a SAML SCT. The detail mapping of these contents to SAML elements will be discussed in Section 4.2.

After the requestor receives the conversation tokens from the STS, it will parse its SAML SCT into a local format that can be recognized by the key store. When the requestor begins a secure conversation with the target, it will encrypt the message using its context token and attach the conversation identifier and the target's SAML SCT as well. The conversation target will authenticate the conversation through the attached tokens and decrypt the message with the context token that is extracted from its SAML SCT using the same algorithm as the requestor.

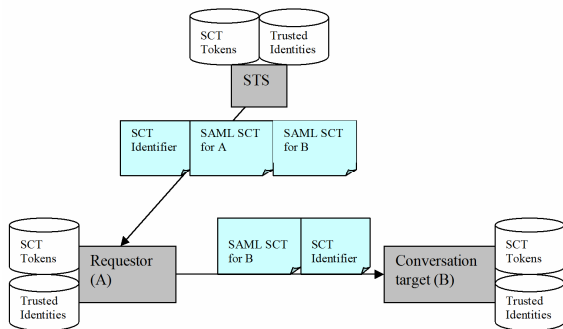


Figure 3. Framework for SAML secure context token based Web services secure conversation

4.2. SAML Secure Context Token

We use the SAML *AuthenticationStatement* assertion to construct our SAML SCTs. Here the subject an STS will authenticate is a conversation. Figure 4 gives the SAML SCT format. We map the core attributes of a secure context into the elements or their attributes of a SAML SCT as follows:

Conversation identifier: the *AssertionID* attribute will represent the conversation identifier generated by the STS and shared by the conversation pair.

Life time: The elements of *NotBefore*, *NotOnOrAfter* and the *IssueInstant* attribute can decide an exact life time for each SCT.

Applies_to: this attribute defines the permitted conversation pairs with the SCT. We use the element of *NameIdentifier* in the *Subject* statement to denote this attribute. For example, if two users A and B begin a conversation, the SAML SCT for A will set this element to "B" and the SAML SCT for B will set this element to "A".

Security token: the security token is encrypted using the receiver's public key and put into the *SubjectConfirmationData* element for each SAML SCT.

We map the authentication related information into the elements of the SAML SCT as follows:

STS identifier: The Issuer attribute is used to denote it.

STS authentication information: The STS can use the *Signature* element to authenticate the assertion. It is interesting to notice that in a federated STS scenario, which means the requestor and the conversation target may trust different STSs, each signature of the two SAML SCTs should be generated by the corresponding (trusted) STS.

Optional authenticated identities: basically, for a secure conversation itself, the above information is enough to establish a trusted secure conversation between two entities. However, sometimes, if a STS involves many SCT renewing procedures, it can encapsulate other information such as the identities of conversation peers into the *KeyInfo* element of a Subject statement, so that the conversation pair can establish a direct trust relationship using the authenticated identities. After that, the renewing procedure can be processed by the conversation pairs directly.

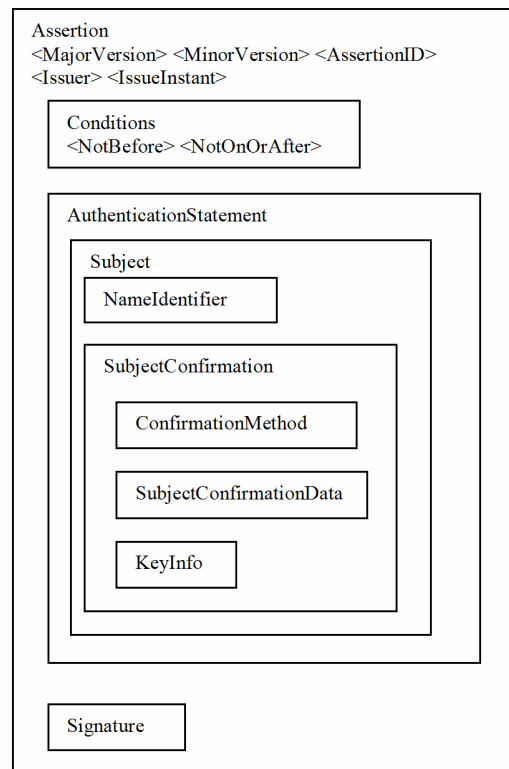


Figure 4. SAML secure context token format

4.3. Authentication of a Secure Conversation

In Figure 3, we see that the authentication process of a secure conversation can be divided into 2 parts: authentication of the SCT from the requestor side and authentication of the SCT from the conversation target side. For the first part, since we assume the requestor trusts the STS directly, it can authenticate the SCT using the STS signature either from the SAML SCT for it or from the underlying message signature, such as WS-Security Signature in terms of Web services implementations. The second part is more complicated than the first part since the conversation requestor and the conversation target do not have a direct trust relationship. We will only discuss the establishment stage here because after this stage, our framework behavior will be the same as many existing systems. For example, both of conversation peers can use the security token included in their SAML SCTs as the master key and derive different keys for integrity and confidentiality usages, like SSL/TLS [12] does. During the establishment stage, on the requestor side, it needs to encrypt the first message using its SCT, and attach the conversation identifier and the SAML SCT of the target to this message. On the conversation target side, when it receives this message, the target needs to authenticate this conversation using 3 operations:

Operation 1: authenticating the SAML SCT using the enclosed STS signature;

Operation 2: verifying the context by checking life time, applies-to in the SAML SCT and the consistence of the conversation identifier with the assertion identifier.

Operation 3: it will decrypt and check the integrity of the message using the key included in the SAML SCT.

Only if all these operations are passed, can the conversation target trust and establish a secure conversation with the requestor.

Through operation 1, we can resist some classical attacks such as modification of the SAML SCT or faking a SAML SCT either by the requestor or by the man in the middle of communication. Through operation 2, we can check the validity of a conversation as well as resist the modification of the conversation identifier. Operation 3 can resist the misuse of a SAML SCT attack. For example, if an attacker gets the SAML SCT for a conversation target from the requestor side, he can fake a SCT as well as a conversation identifier and begin a secure conversation with the target as the actual requestor does above. However, since the conversation security token is encrypted by the target's secret key (like the public key in the asymmetric algorithms) in the SAML SCT, the attacker can not construct the same security token. Thus, the conversation target can always detect this attack through decryption and message integrity check.

5. Web Services Standards Based Framework Implementation and Application to a Grid Monitoring System

We have implemented the core components of SAML based Web Services secure conversation framework on the Java platform with PKI. Our implementation is conformant to the emerging Web services standards including WS-Security [3], WS-Security SAML token profile [15], WS-Trust [1] and WS-SecureConversation [2]. As an application, we describe how to use it to secure a Grid monitoring system that may involves many secure conversations.

5.1. Web Services Standards Based Implementation Approach

Basically, our implementation is based on Figure 3, and integrates the related Web services standards into it as shown in Figure 5. For example, if a requestor M and a conversation target N will establish a Web service based secure conversation through a STS T, the sample tokens response message from T is showed in Appendix 2, and the establishment message from A to B is showed in Appendix 3. We can see the WS-Trust, WS-SecureConversation, SAML and WS-Security SAML token profile can be easily integrated into our framework.

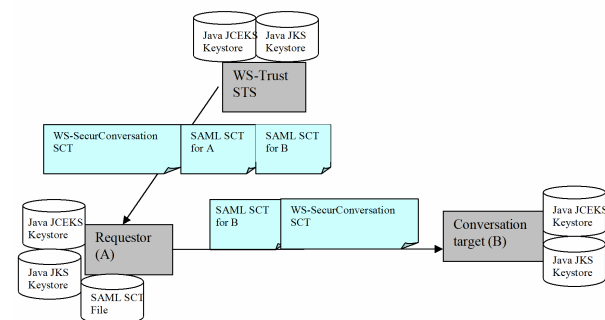


Figure 5. Implementation Framework of SAML secure context token based Web services secure conversation

5.2. Token Management

In our framework, there are three types of tokens: PKI related X509V3 tokens, SAML SCT, and SCT. X509V3 tokens are stored as JKS format that is encrypted using passwords. A SCT is the serialization object of a secure context class and is stored as JCEKS format that can be the encrypted using passwords as well. In Figure 5, we notice that the requestor needs to store temporarily and forward the SAML SCT to the conversation target. Since the security token of a SAML SCT is encrypted by the conversation target's public key, and the SAML SCT is signed by the STS, we can store all SAML SCTs into an

XML file directly. Thus, we can save the processing time of translation from a SAML SCT to a SCT and reverse.

5.3. A Grid Monitoring Architecture Based Monitoring System

The Grid Monitoring Architecture (GMA) [13] is a framework draft from GGF that is specifically designed to fit the requirements of monitoring in Grid environments. It gives an abstract account of the core components that should constitute a Grid monitoring system and their interaction protocols. There are three main roles for GMA entities, the producer, consumer and directory service. Producers send event data to consumers using Push (Subscribe/Notify) or Pull (Query/Response) models. Both producers and consumers can use a directory service for publication and discovery. Figure 6 depicts the 3 roles and their interaction patterns as defined in GMA.

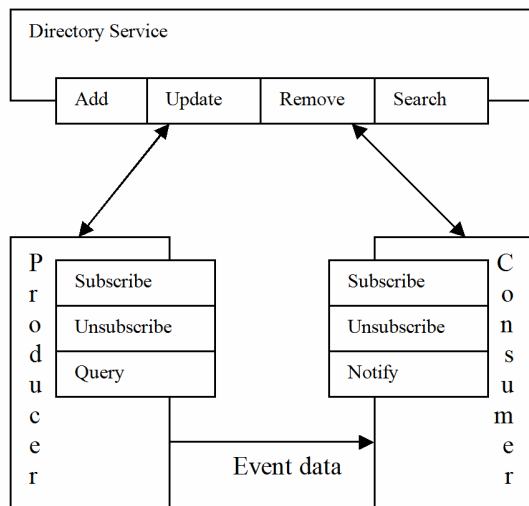


Figure 6. GMA Roles, Functionalities, Interactions

In terms of Web Services, the emerging standards of Web Services Notification [14] defines an event driven framework along with some related Web Services portals that can be used to monitor events. We have implemented a WS-Notification based GMA system called WSN-GMA that can run in standard Web services based Java hosting environment [16].

5.4. Application of SAML Based Secure Conversations to Secure GMA

As introduced in Section 5.3, the producers can discover some consumers based on their event interest matching procedure, and send some events directly to the consumers. In our application of WSN-GMA to a federated Web services IDS, we find that each producer and each consumer must trust a directory service either directly or through federated approaches such as bridged

CAs [17] or Web services federations [8,11], but they do not necessarily trust each other directly. We have implemented the secure conversation establishment between a producer and a consumer without a direct trust relationship using the IDTSC issue-forward-use type. Here the directory service acts as a WS-Trust STS. When a producer or a consumer sends a *Search* request to the directory service, the directory service will generate the necessary tokens as described in Section 4.1 for each search result (the potential conversation pair), and return them with the search results to the requestor. Appendix 1 gives a sample XML message that encapsulates the tokens into the search results.

6. Discussion

We now discuss some possible performance improvements of the implementation. As showed in Section 5.4, if our framework is encapsulated into other messages, such as WS SOAP messages, then in order to decrease the encryption and signature operations from the STS side, we can replace the SAML SCT for the requestor by the corresponding WS-Trust elements for life time, applies-to and security token. In this way we can reuse the WS-Security operations for encryption and signature that are involved in a SAML SCT. We cannot replace the SAML SCT for the conversation target by the WS-Trust elements because the requestor needs to forward it afterwards.

One modification of the issue-forward-use type is to separate the establishment of a direct trust relationship from the establishment of a secure context between two entities. First, the STS can include only the authenticated identities of the conversation pair into the SAML tokens, and send them to the requestor. For example, in the PKI implementation, the STS will put the certificate of the requestor into the SAML Authentication assertion for the conversation target and include the conversation target's certificate into the SAML assertion for the requestor. The requestor and the conversation target can mutually authenticate using the SAML tokens. After that, the requestor and the conversation target can negotiate and establish a secure conversation between themselves directly using this trust relationship, thereby migrating the work load of establishment from the STS to the conversation pairs.

7. Conclusion

In this paper, we have investigated the different secure conversation frameworks according to their establishment procedures. Focusing on those secure conversations with indirect trust relationships, we describe some important scenarios that require a new establishment type called issue-forward-use, which has not been reported in previous works. We present a SAML based

approach to support this type. A trustable establishment protocol is analyzed from both the normal authentication viewpoint and the attacks viewpoint. The different modifications of our framework are adopted to different performance and application requirements. The implementation of our framework using the emerging Web services standards and its application to secure a Grid monitoring system shows our generic framework can be easily used to support Web services based secure conversation.

8. Acknowledgements

The author is thankful to Luigi Lo Iacono, Gregory A. Kohring, Guy Lonsdale, and Jochen Fingberg, the colleagues from the NEC C&C Research Laboratories, for helping the author to clarify the ideas and correct the grammar errors in this work. This work is supported in part by the NextGrid project (contract number 511563), which is funded by the European Commission's IST 6th Framework Programme. .

9. References

- [1] WS-Trust V1.0 Working Draft, OASIS Web Services Secure Exchange TC, <http://www.oasis-open.org/committees/download.php/16138/oasis-wssx-ws-trust-1.0.pdf>, 2006
- [2] WS-SecureConversation V1.0 Working Draft, OASIS Web Services Secure Exchange TC, <http://www.oasis-open.org/committees/download.php/16140/oasis-wssx-ws-secureconversation-1.0.pdf>, 2006
- [3] WS-Security Standard V1.0, OASIS Web Services Security TC, <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>, 2004
- [4] Security Assertion Markup Language http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security
- [5] D. Smith, "WS-Security Drilldown in Web Services Enhancements 2.0", MSDN, 2004
- [6] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke. Security for Grid Services. *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, IEEE Press, June 2003.
- [7] S. Middleton et al. GEMSS: Privacy and security for a Medical Grid. In *Proceedings of HealthGrid 2004*, January 2004.
- [8] Liberty ID-FF Architecture Overview, Liberty Alliance Project ID-FF 1.2 Final, <http://www.projectliberty.org/specs/draft-liberty-idff-arch-overview-1.2-errata-v1.0.pdf>
- [9] J. Wang, D. Del Vecchio, and M. Humphrey. Extending the Security Assertion Markup Language to Support Delegation for Web Services and Grid Services. *IEEE International Conference on Web Services (ICWS-2005)*, IEEE Press, July 2005.
- [10] Assertions and Protocol for the OSAIS SAML 1.1. 2003. <http://www.oasis-open.org/committees/download.php/3406/oasis-%20sstc-saml-core-1.1.pdf>
- [11] Web Services Federation Language, July 2003
- [12] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, 2000
- [13] B. Tierney et al. A Grid Monitoring Architecture, GGF Informational, <http://www.gridforum.org/documents/GFD.7.pdf>, 2002
- [14] Web Services Notification, OASIS Web Services Notification TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn
- [15] WS-Security SAML Token Profile V1.0, OASIS Web Services Security TC, <http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.0.pdf>, 2004
- [16] Apache Axis Project, <http://ws.apache.org/axis/>
- [17] J. Jokl, J. Basney, and M. Humphrey. [Experiences using Bridge CAs for Grids](#). *Proceedings of the UK Workshop on Grid Security Experiences*. Oxford 8th and 9th July 2004.

Appendix

1. A sample message showing encapsulation of SAML context tokens into GMA search results

```

<?xml version="1.0" encoding="UTF-8"?><soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsa:Action soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">http://schemas.xmlsoap.org/ws/2004/08/addressing/anonymous
  </wsa:Action>
    <wsa:To soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">http://schemas.xmlsoap.org/ws/2004/08/addressing/anonymous
  </wsa:To>
    <IssuedTokens soapenv:actor="" soapenv:mustUnderstand="0" xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">
      <RequestSecurityTokenResponse xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust"> } Tokens for Search Result1
      ...
    </RequestSecurityTokenResponse>
    <RequestSecurityTokenResponse xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust"> } Tokens for Search Result2
    ...
  </RequestSecurityTokenResponse>
</IssuedTokens>
</soapenv:Header>
<soapenv:Body>
  <not:SearchResponse xmlns:not="http://ws.apache.org/brokerednotification/base/service/NotificationBroker">
    <not:Match> } Result1
    ...
  </not:Match>
  <not:Match> } Result2
    ...
  </not:Match>
</not:SearchResponse>
</soapenv:Body>
</soapenv:Envelope>

```

2. A sample response message from the STS

```

<RequestSecurityTokenResponse xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">
  <RequestedSecurityToken xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">
    <SecurityContextToken Id="sct-67q990s3" xmlns="http://schemas.xmlsoap.org/ws/2005/02/sc">
      <Identifier xmlns="http://schemas.xmlsoap.org/ws/2005/02/sc">75e1f7ed-ccbc-4e06-bc11-45315717d286</Identifier> } WS-SecureConversation
    </SecurityContextToken> } SCT
    <RequestedSecurityToken>
      <RequestedProofToken xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">
        <Assertion AssertionID="Assertion-75e1f7ed-ccbc-4e06-bc11-45315717d286" IssueInstant="2006-01-20T15:01:33.941Z"
Issuer="T" MajorVersion="1" MinorVersion="1" xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
          <Conditions NotBefore="2006-01-20T15:01:33.940Z" NotOnOrAfter="2006-01-22T15:01:33.940Z"
xmlns="urn:oasis:names:tc:SAML:1.0:assertion"/>
          <AuthenticationStatement AuthenticationInstant="2006-01-20T15:01:33.940Z" AuthenticationMethod="X509V3"
xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
            <Subject xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
              <NameIdentifier xmlns="urn:oasis:names:tc:SAML:1.0:assertion">B </NameIdentifier>
              <SubjectConfirmation xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
                <ConfirmationMethod xmlns="urn:oasis:names:tc:SAML:1.0:assertion">urn:oasis:names:tc:SAML:1.0:cm:holder-of-
key</ConfirmationMethod>
                <SubjectConfirmationData xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
                  <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
                    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
                      <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
                        <xenc:CipherValue> ... </xenc:CipherValue>
                      </xenc:CipherData>
                    </xenc:EncryptedKey>
                  </SubjectConfirmationData>
                  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                    <ds:KeyName xmlns:ds="http://www.w3.org/2000/09/xmldsig#">N_Cert_Name</ds:KeyName>
                    <ds:X509Data xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                      <ds:X509Certificate xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                        ...
                      </ds:X509Certificate>
                    </ds:X509Data>
                  </ds:KeyInfo>
                </SubjectConfirmation>
              </Subject>
            </AuthenticationStatement>
            <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
              ...
            </ds:Signature>
          </Assertion>
        </RequestedProofToken>
      </RequestedProofToken xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">
        <Assertion ...>
          ...
        </Assertion>
      </RequestedProofToken>
    </RequestSecurityTokenResponse>

```

3. A sample message for establishing a secure conversation

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wsssecurity-secext-1.0.xsd">
      <Assertion AssertionID="Assertion-75e1f7ed-ccbc-4e06-bc11-45315717d286" IssueInstant="2006-01-20T15:01:33.941Z"
Issuer="T" MajorVersion="1" MinorVersion="1" xmlns="urn:oasis:names:tc:SAML:1.0:assertion"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion" xmlns:samlp="urn:oasis:names:tc:SAML:1.0:protocol">
        ...
      </Assertion>
      <sc:SecurityContextToken oas:Id=" sct-67q990s3 " xmlns:sc="http://schemas.xmlsoap.org/ws/2005/02/sc"
xmlns:oas="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsssecurity-utility-1.0.xsd"><sc:Identifier>75e1f7ed-ccbc-
4e06-bc11-45315717d286</sc:Identifier></sc:SecurityContextToken>
      <xenc:ReferenceList><xenc:DataReference URI="#EncDataId-12254719"/></xenc:ReferenceList></wsse:Security>
      <wsa:Action soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">http://schemas.xmlsoap.org/ws/2004/08/addressing/anonymous
</wsa:Action>
      <wsa:To soapenv:actor="http://schemas.xmlsoap.org/soap/actor/next" soapenv:mustUnderstand="0"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">http://schemas.xmlsoap.org/ws/2004/08/addressing/anonymous
</wsa:To>
    </soapenv:Header>
    <soapenv:Body><xenc:EncryptedData Id="EncDataId-12254719"
Type="http://www.w3.org/2001/04/xmlenc#Content"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/><ds:KeyInfo
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsssecurity-secext-
1.0.xsd"><wsse:Reference URI=" sct-67q990s3"
ValueType="http://schemas.xmlsoap.org/ws/2005/02/sc/scet"/></wsse:SecurityTokenReference>
    </ds:KeyInfo><xenc:CipherData><xenc:CipherValue> ...</xenc:CipherValue></xenc:CipherData></xenc:EncryptedData></soa
penv:Body>
  </soapenv:Envelope>

```

SAML SCT
for B

WS-SecureConversion
SCT

WS-Security
Encryption