GWD-R.P (submitted for consideration)
GGF OGSA Security Workgroup

Samuel Meder
Frank Siebenlist
Von Welch
Jarek Gawor
Thomas Sandholm
Argonne National Laboratory
February, 2003
Revised 2/15/2003

A GSSAPI profile for security context establishment and message protection using WS-SecureConversation and WS-Trust

Status of this Memo

This document has been submitted to the Global Grid Forum OGSA Security Working Group for consideration as a Recommendation (GWD-R).

The latest version of this document can be found at:
http://www.globus.org/ogsa/Security/

Copyright Notice

1. Abstract

This document describes how a secure association can be established between OGSA/Web Services peers using a GSSAPI-mechanism and how this association can be used for subsequent message protection. Port types and message formats are defined for context establishment, and procedures and conventions are defined for per-message protection. The GSSAPI-mechanism provides the authentication, QoP-negotiation/agreement and actual signing and encryption operations. WS-SecureConversation and WS-Trust are used to frame the GSSAPI security context establishment. XML-Signature and XML-Encryption are employed to provide for the associated transformations and processing of for the message protection. A standard WS-Security compliant header element is used to communicate the relevant security information, i.e. XML-Signature and XML-Encryption elements, with the protected message to facilitate easy validation and decryption of the message.

No constraints are placed on the tokens emitted by the GSSAPI-mechanism, enabling this specification to be used with arbitrary GSSAPI-mechanisms. Also, the described scheme does not modify standard XML-Signature and XML-Encryption processing, allowing easy integration with XML-Signature and XML-encryption implementations that feature pluggable signature and encryption algorithms.

Table of Contents

2. Introduction

The GSSAPI provides a standard API for establishing a security context that can be implemented using an arbitrary underlying security mechanism. Implementations of the GSSAPI currently exist and are in widespread use for Kerberos [1] and the Grid Security Infrastructure (GSI).

Web services security standards currently provide no well-defined method for establishing a security session. This document is a specification for leveraging the GSSAPI to perform security session establishment and message protection in web services, allowing existing GSSAPI implementations to be used in a Web Services framework.

The WS-SecureConversation and WS-Trust documents are used to provide a framework for exchanging the security tokens generated during GSSAPI security context establishment. The XML-Signature, XML-Encryption and WS-Security standards are used to define GSSAPI based per-message protection for web services.

3. Notations and Terminology

This section specifies the notations, namespaces, and terminology used in this specification.

3.1 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [3].

3.2 Namespace

The XML namespace URI that MUST be used by implementations of this specification is:

```
http://www.globus.org/ws/2003/2/secext-gss
```

The following namespaces are used in this document:

Prefix	Namespace
ds	http://www.w3.org/2000/09/xmldsig#
gss	http://globus.org/ws/2003/2/secext
wsse	http://schemas.xmlsoap.org/ws/2002/12/secext
wsu	http://schemas.xmlsoap.org/ws/2002/07/utility
xenc	http://www.w3.org/2001/04/xmlenc#
xsd	http://www.w3.org/2001/XMLSchema

4. Context Establishment

The GSSAPI context establishment is initiated by a call to gss_init_sec_context() which emits a context token for consumption by the accepting side (gss_accept_sec_context()). The accepting side may in turn respond with a token for consumption by the initiating side. Subsequent calls to gss_init_sec_context() and gss_accept_sec_context() will consume and produce tokens until either an error occurs or a security context is established. The GSSAPI specification relies on other mechanisms for transporting these security tokens between the parties involved in the establishment. This section specifies a binding independent message format for transporting security tokens and the WSDL for a security session establishment porttype.

4.1 Initial Context Token

The initial context token is wrapped in the following way:

```
<RequestSecurityToken xmlns="http://schemas.xmlsoap.org/ws/2002/12/secext>
        <TokenType>gssapi-context</TokenType>
        <RequestType>wsse:ReqExchange</RequestType>
        <gss:MechanismType> GSSAPI Mechanism OID </gss:MechanismType>
        <gss:Continue> True/False </gss:Continue>
        <wsse:SecurityContextToken>
                <wsu:Identifier> uuid:Context Identifier </wsu:Identifier>
        </wsse:SecurityContextToken>
        <BinaryNegotiation ValueType="gssapi-context-token"
                        EncodingType="wsse:Base64Binary">
                Base 64 Encoded GSSAPI Context Token
        </BinaryNegotiation>
</RequestSecurityToken>
```

Implementers MUST take care to ensure that the context identifier (uuid:Context Identifier) is unique to both initiator and acceptor and that it can be utilized as a ds:Keyname element.

In the above the MechanismType and Continue elements are defined as:

```
<xsd:schema targetNamespace="http://globus.org/ws/2003/2/secext"
                xmlns="http://globus.org/ws/2003/2/secext" >
        <xsd:element name="MechanismType" type="xsd:string"/>
        <xsd:element name="Continue" type="xsd:boolean" optional=true/>
</xsd:schema>
```

The MechanismType element MUST indicate the security mechanism used to generate the initial context token. The MechanismType value can be used by parties to match and publish their

supported mechanisms. Furthermore, when the runtime supports multiple different GSSAPI mechanisms, then the MechanismType value will allow the runtime to dispatch the initial context establishment token to the correct GSSAPI implementation.It is assumed that all subsequent tokens will be generated using said mechanism. The runtime should be able to dispatch subsequent tokens correctly based on the SecurityContextToken/Identifier value.
If present, tContinue element MUST indicate whether the token producer requires more information, i.e. tokens, to establish the context. A Continue element value of “false” can be used by the runtime to decide whether the application message could be submitted for protection by the locally established context, and send together with the context establishment message.

4.2 Subsequent Context Tokens

After the initial context token further negotiation is by exchange of messages of the type:

```
<RequestSecurityTokenResponse xmlns="http://schemas.xmlsoap.org/ws/2002/12/secext">
        <TokenType>gssapi-context</TokenType>
        <gss:Continue> True/False </gss:Continue>
        <RequestedSecurityToken>
                <wsse:SecurityContextToken>
                        <wsu:Identifier> uuid:Context Identifier </wsu:Identifier>
                </wsse:SecurityContextToken>
                <BinaryNegotiation ValueType="gssapi-context-token"
                                EncodingType="wsse:Base64Binary">
                        Base 64 Encoded GSSAPI Context Token
                </BinaryNegotiation>
        </RequestedSecurityToken>
</RequestSecurityTokenResponse>
```

The “Continue” element is defined as in section 4.2. The context identifier ("uuid:Context Identifier") MUST be identical to the one sent with the initial context token, i.e. the one found in the RequestSecurityToken element.

4.3 Context Establishment Port Type

This section defines the porttype and associated messages for a security context establishment service. The port type defines two operations: initContext and establishContext. The initContext operation consists of the initial message exchange, i.e. a RequestSecurityToken element answered by a RequestSecurityTokenResponse. All subsequent message exchanges MUST use the establishContext operation.

```
<message name="InitContextInputMessage">
        <part name="parameters" element="wsse:RequestSecurityToken"/>
</message>

<message name="InitContextOutputMessage">
        <part name="parameters" element="wsse:RequestSecurityTokenResponse"/>
</message>

<message name="EstablishContextInputMessage">
        <part name="parameters" element="wsse:RequestSecurityTokenResponse "/>
</message>

<message name="EstablishContextOutputMessage">
        <part name="parameters" element="wsse:RequestSecurityTokenResponse"/>
```

```
</message>

<portType name="ContextEstablishmentPortType">
        <operation name="initContext">
                <input message="tns:InitContextInputMessage"/>
                <output message="tns:InitContextOutputMessage"/>
        </operation>
        <operation name="establishContext">
                <input message="tns:EstablishContextInputMessage"/>
                <output message="tns:EstablishContextOutputMessage"/>
        </operation>
</portType>
```

Note that none of the XML element and attribute values are cryptographically protected. The GSSAPI-tokens have their own security built-in to their tokens and protocols, so the assumption is that this will be sufficient for protecting the exchange.

5. Per-Message Protection

5.1 Signing

The GSSAPI provides two methods for signing a message. The first, gss_wrap, outputs a token that includes both the signature and the data that is being protected and is thus not suitable for integration with the XML-Signature specification. The second, gss_getMIC(), emits a token containing only the signature and is thus suitable for use with the XML-Signature standard.

The XML-Signature runtime is used to provide a digest of the message. In stead of the complete message, only this digest value will be used for the input for the gss_getMIC() call. This allows us to leverage the standard XML-Signature transformation and processing.

Per-message signing is accomplished using the gss_getMIC() call. The output of the gss_getMIC() call MUST be framed in the following way:

```
<wsse:Security>
        <wsse:SecurityContextToken wsu:Id="MyContext">
                <wsu:Identifier> uuid:ContextIdentifier </wsu:Identifier>
        </wsse:SecurityContextToken>

        <ds:Signature>
                <ds:SignedInfo>
                        <ds:CanonicalizationMethod
                                Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
                        <ds:SignatureMethod
                                Algorithm="http://www.globus.org/2002/04/xmldsig#gss-sign"/>
                        <ds:Reference URI="#digestSource">
                                <ds:DigestMethod
                                        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                                <ds:DigestValue>
                                                1CnwAoXjx6FRDJ7y/Z0e81yB2yA=
                                </ds:DigestValue>
                        </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
```

```
                    signature token emitted by gss_get_MIC()
                </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#MyContext"/>
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>
        </ds:Signature>
</wsse:Security>
```

TTInfothe KeyName element MUST refers to a SecurityContextToken in the Security header element, which in turn MUST refers to a previously established security context through its Identifier element..
Note that the signing algorithm identifier is used to enable the XML-Signature runtime to invoke the GSSAPI mechanism for the signing operation.
Also, future versions of this draft should provide a mechanism for specifying or at the very least allow for digest algorithms other than SHA1. The above example shows the use of SHA1 as the digest algorithm. Note, however, that the GSSAPI runtime will use its own digest algorithm on the XML-Signature generated digest, which is determined by the GSSAPI-mechanism's own capabilities and policy.

5.2 Encryption

Encryption MUST be done using the gss_wrap() call with appropriate options for encryption. This will yield a token that encapsulates the input user data along with associated integrity check. This token MUST be framed using the WS-Security and XML-Encryption standards. Assuming a SOAP binding this would result in a SOAP header containing:

```
<wsse:Security>
        <wsse:SecurityContextToken wsu:Id="MyContext">
            <wsu:Identifier>uuid:ContextIdentifier</wsu:Identifier>
        </wsse:SecurityContextToken>

        <xenc:ReferenceList>
            <xenc:DataReference URI="#elementID"/>
        </xenc:ReferenceList>
</wsse:Security>
```

A SOAP body with the following element would follow this header:

```
<xenc:EncryptedData Id="elementID">
        <xenc:EncryptionMethod
            Algorithm="http://www.globus.org/2002/04/xmlenc#gss-enc"/>
        <ds:KeyInfo>
            <wsse:SecurityTokenReference>
                <wsse:Reference URI="#MyContext"/>
            </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <xenc:CipherData>
            <xenc:CipherValue>
                encrypted token emitted by gss_wrap()
            </xenc:CipherValue>
        </xenc:CipherData>
</xenc:EncryptedData>
```

The KeyInfo element again MUST refers to a SecurityContextToken in the Security header element, which in turn MUST refers to a previously established security context through its Identifier element.
Note that the encryption algorithm identifier in the EncryptionMethod element is used to enable the XML-Encryption runtime to invoke the GSSAPI mechanism for the encryption operation.

6. Considerations

To ease the burden of processing large header when messages go through multiple intermediaries with their own security contexts one could consider tagging each WS-Security header element with a target designator. There MUST be one such WS-security header element for each receiver/intermediary/security-name.

This document does not address any error processing concerns. In general it is our opinion that errors currently specified in the WS-* specifications should be sufficient for dealing with most basic error conditions and that richer errors may be propagated in the form of GSSAPI security tokens.

The use of GSSAPI for signature and encryption removes the ability for XML based tools to inspect a document for the level of protection used, i.e. it becomes impossible to determine what algorithms were used in e.g. signing a element.

A feature that is currently missing form the security session establishment port type described herein is the ability to delete existing security contexts. An operation allowing this functionality may be added at a later time.

Note that this specification is based on the initial published version of WS-SecureConversation and WS-Trust, which at the time of writing of this document, have not been submitted to any standards organization. Furthermore, these initial WS-* version are clearly drafts with no concise schema definitions, and it is expected that their content will change as they progress throughthrough the scrutiny of the standardizations processcommunity.

Unfortunately, we had to define a number of new xml-elements to render a GSSAPI mechanism on the WS-Trust framework (gss:MechanismType, gss:Continue), and we had to use elements in ways that are not specified in the current draft (use wsse:SecurityContextToken as a sub-element in a RequestSecurityToken element), and we had to defined a new portType because those in the current spec didn't seem to fit the semantics well. Hopefully, we will be able tocan convince the committee in charge ofthat will standardizing thee WS-Trust specification to adopt changes that would facilitate a clean GSSAPI rendering without the definition of new elements and semantics.

Our initial design and implementation used a context identifier value that was contributed by both initiator and acceptor as is suggested in SPKM [11]. The current WS-Trust spec does not provide the facility to implement this. We hope, however, to find a way to incorporate this feature in future versions of this spec and to convince standardizing committee of its importance.

7. Glossary

GFD:	Grid Forum Document. GFDs are persistent.
GGF:	Global Grid Forum. See www.gridforum.org
GWD:	Grid Working Document. GWDs are not persistent, but exist as drafts for discussion.

8. Authors Contact Information

Samuel Meder (meder@mcs.anl.gov)
Frank Siebenlist (franks@mcs.anl.gov)
Jarek Gawor (gawor@mcs.anl.gov)
Thomas Sandholm (sandholm@mcs.anl.gov)

Argonne National Laboratory
9700 S. Cass Avenue
Argonne, IL 60439-4844

Von Welch (welch@mcs.anl.gov)

Distributed Systems Laboratory
University of Chicago and Argonne National Laboratory

9. Acknowledgements

We are grateful to numerous colleagues for discussions on the topics covered in this paper, in particular (in alphabetical order, with apologies to anybody we've missed): Nataraj (Raj) Nagaratnam, and Steve Tuecke.

This work was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract …; by the IBM Corporation?

10. Intellectual Property Statement

11. Full Copyright Notice

12. References

[1] Kohl, J. and C. Neuman, "The Kerberos Network Authentication Service (V5)," RFC 1510, September 1993.

[2] Dierks, T. and C. Allen, "The TLS Protocol, Version 1.0," RFC 2246, January 1999.

[3] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997.

[4] Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal of Supercomputer Applications, 2001.

[5] Linn, J., "Generic Security Service Application Program Interface, Version 2, Update 1," RFC 2743, January 2000.

[6] Wray, J., "Generic Security Service API Version 2, C-bindings," RFC 2744, January 2000.

[7] Wray, J., "Generic Security Service API: C-bindings", RFC 1509, September 1993.

[8] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.

[9] Adams, C., "The Simple Public-Key GSS-API Mechanism (SPKM)", RFC 2025, October 1996.

[10] D. Eastlake, J. Reagle, and D. Solo, XML-Signature Syntax and Processing. IETF Draft Standard/W3C Recommendation, August 2001.

[11] D. Beech, M. Maloney, N. Mendelsohn, and H. Thompson, XML Schema Part 1: Structures. W3C Recommendation, May 2001.

[12] P. Biron, A. Malhotra, XML Schema Part 2: Datatypes. W3C Recommendation, May 2001.

[13] FIPS PUB 180-1. Secure Hash Standard. U.S. Department of Commerce/National Institute of Standards and Technology.

[14] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Frystyk Nielsen, S. Thatte, and D. Winer, Simple Object Access Protocol (SOAP) Version 1.1, W3C Note, May 2001.

13. Appendix: WSDL for SecureContextEstablishmentPortType

The below gives the WSDL for the context establishment port type. It should be noted that this description does not declare any fault messages and that fault messages based on the ones described in the WS-Trust and WS-SecureConversation specifications should be added.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="SecurityContextEstablishmentDefinition"
            targetNamespace="http://ogsa.globus.org/security/authentication"
            xmlns:tns=http://ogsa.globus.org/security/authentication
            xmlns:wsse= http://schemas.xmlsoap.org/ws/2002/12/secext
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://schemas.xmlsoap.org/wsdl/">

<import location="http://schemas.xmlsoap.org/ws/2002/12/secext"
       namespace=" http://schemas.xmlsoap.org/ws/2002/12/secext"/>

<message name="InitContextInputMessage">
  <part name="parameters" element="wsse:RequestSecurityToken"/>
</message>

<message name="InitContextOutputMessage">
  <part name="parameters" element="wsse:RequestSecurityTokenResponse"/>
</message>

<message name="EstablishContextInputMessage">
  <part name="parameters" element="wsse:RequestSecurityTokenResponse"/>
</message>

<message name="EstablishContextOutputMessage">
  <part name="parameters" element="wsse:RequestSecurityTokenResponse"/>
</message>

<portType name="SecureContextEstablishmentPortType">

  <operation name="initTokenExchange">
    <input message="tns:InitTokenExchangeInputMessage"/>
    <output message="tns:InitTokenExchangeOutputMessage"/>
  </operation>

  <operation name="continueTokenExchange">
    <input message="tns:ContinueTokenExchangeInputMessage"/>
    <output message="tns:ContinueTokenExchangeOutputMessage"/>
  </operation>

</portType>

</definitions>
```

14. Appendix: SOAP Protected Message Example

The following example shows a SOAP header and body, where the body content is signed. The signature information is communicated through a Signature element inside of a WS-Security header element.
The elements and attributes to note are:

- the KeyName element value, which allows us to identify the security context
- the SignatureMethod attribute, which indicates that the “algorithm” is in fact a GSSAPI mechanism,
- the SignatureValue, which is the base64 encoded gss-token that holds the signature.

```
<SOAP-ENV:Envelope SOAP-ENV:actor="some-uri" SOAP-ENV:mustUnderstand="1"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

    <SOAP-ENV:Header>
        <wsse:Security>
            <wsse:SecurityContextToken wsu:Id="MyContext">
                <wsu:Identifier>1234567</wsu:Identifier>
            </wsse:SecurityContextToken>
            <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
                <ds:SignedInfo>
                    <ds:CanonicalizationMethod
                       Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                    <ds:SignatureMethod
                       Algorithm="http://www.globus.org/2002/04/xmlenc#gss-getmic"/>
                    <ds:Reference URI="#digestSource">
                    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                    <ds:DigestValue>
                        1CnwAoXjx6FRDJ7y/Z0e81yB2yA=
                    </ds:DigestValue>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
                    439ajDhD6VD+tU1fh++01Q==
                </ds:SignatureValue>
                <ds:KeyInfo>
                    <wsse:SecurityTokenReference>
                        <wsse:Reference URI="#MyContext"/>
                    </wsse:SecurityTokenReference>
                </ds:KeyInfo>

                </ds:KeyInfo>
            </ds:Signature>
        </wsse:Security>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <getValue id="digestSource"
           xmlns="http://samples.ogsa.globus.org/counter/counter_port_type"/>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```