Secure Password-Based Authenticated Key Exchange for Web Services

Liang Fang[1]
Computer Science Department, Indiana University
lifang@cs.indiana.edu

Samuel Meder
Department of Computer Science, University of Chicago
meder@mcs.anl.gov

Olivier Chevassut
Computational Research Division, Lawrence Berkeley National Laboratory
ochevassut@lbl.gov

Frank Siebenlist
Mathematics and Computer Science Division, Argonne National Laboratory
franks@mcs.anl.gov

ABSTRACT
This paper discusses an implementation of an authenticated key-exchange method rendered on message primitives defined in the WS-Trust and WS-SecureConversation specifications. This IEEE-specified cryptographic method (AuthA) is proven-secure for password-based authentication and key exchange, while the WS-Trust and WS-SecureConversation are emerging Web Services Security specifications that extend the WS-Security specification. A prototype of the presented protocol is integrated in the WS-ResourceFramework-compliant Globus Toolkit V4. Further hardening of the implementation is expected to result in a version that will be shipped with future Globus Toolkit releases. This could help address the current unavailability of decent shared-secret-based authentication options in the Web Services and Grid world. Future work will be to integrate One-Time-Password (OTP) features in the authentication protocol.

Categories and Subject Descriptors
C.2.0 [**Computer-Communication Networks**]: General – *security and protection.*

General Terms
Security, Design

Keywords
Authenticated key exchange, web services, password, security.

1. INTRODUCTION

1.1 Grid Computing and Web Services
The term "Grid" refers to systems and applications that integrate and manage resources and services distributed across multiple control domains [9]. Pioneered in an e-science context, Grid technologies are also generating interest in industry as a result of their apparent relevance to commercial distributed-computing applications [7]. The results of this research have been incorporated into a widely used software system called the Globus Toolkit® (GT) [8] that uses public key technologies to address issues of single sign-on, delegation, and identity. The Grid Security Infrastructure (GSI) is the name given to the portion of the Globus Toolkit that implements security functionality.

The recent definition of the Web Service Resource Framework (WSRF) specification and other elements of the Open Grid Services Architecture (OGSA) within OASIS and the Global Grid Forum (GGF) introduce new challenges and opportunities for Grid security [21, 24]. In particular, integration with Web services and hosting environment technologies introduces opportunities to leverage emerging security technologies such as described in the WS-Security, WS-Trust and WS-Secure-Conversation specifications [18, 22, 23].

1.2 Security in Grid Computing and Web Services
Security in Web Services (WS) is still immature in many ways as a great number of emerging specifications seem to be competing and in flux. Recently, however, the basic underpinnings for SOAP message security have been defined by the standardized WS-Security specifications in OASIS [23]. WS-Trust and WS-SecureConversation are proposed extensions of the WS-Security specification, defining message primitives and interfaces for security context establishment, sharing, and session key derivation [18, 22]. Although these specifications have not yet been standardized, the associated authors have publicly stated their intentions in that direction.

Security in Grid Computing is of utmost importance. Any Grid site deployment must provide the basic security mechanisms including authentication, authorization and secure communications. The Grid Security Infrastructure (GSI) component in Globus plays the central role in providing these mechanisms, as well as the extended ones such as single sign-on,

ACM Workshop on Secure Web Services, October 29, 2004, Fairfax VA, USA.

[1] Fang's work was completed during his internship in Mathematics and Computer Science Division, Argonne National Laboratory.

delegation and mutual authentication using public key cryptography [14]. GSI was initially built upon the Transport Layer Security (TLS) protocol, and was enhanced to provide message level authentication, key exchange, data protection, and delegation through the use of proxy certificates.

The Globus Toolkit is being rendered on the web service protocols. For that reason, GSI requires a TLS-like message level protocol built from the WS-security primitives, to support the establishment of a security context between two parties and enable future communications without having to perform an expensive exchange of security credential each time [11]. GSI-SecureConversation is the first effort of such a protocol, which defines its own session-based security mechanism, similarly to WS-SecureConversation. Its implementation is based on public key authentication and has no support for authentication based on shared secrets.

The WS-SecureConversation specification accommodates multiple authentication mechanisms (e.g., username/password, certificates, and capabilities). Unfortunately, there is currently no open source implementation of WS-SecureConversation supporting password-based authentication.

Currently, we see a trend where many Grid site deployments use or plan to use password authentication to obtain public key credentials from a credential service. The recent compromises of user and server machines are resulting in site security policy changes where long-term secrets are no longer to be stored on the user's machines. Instead, long-term credentials will be stored on servers in data centers where their integrity can be better protected. Users will authenticate with a (one-time) password to these credential servers. After successful authentication, the user will obtain short-lived credentials, such as a short-lived X.509 certificate or proxy-certificate, which can subsequently be used for the access of other services on the Grid [15]. This has renewed interest in password-based authentication mechanisms.

MyProxy is such a credential management service that is used at the core of a number of Computing Center's credential repository for its Grid users [10]. This service provides a convenient means for the storage of user credentials (i.e. their X.509 certificates and associated private keys), which can then be retrieved by the user using a traditional username and password interface. In addition to solving a number of usability problems for users, a strong argument can be made that consolidating long-term user credentials in a secure repository, such as MyProxy, which is managed by professional staff, is preferable to having credentials managed in an ad hoc fashion by users who are well-intended, but often not security savvy.

1.3 Accomplishments

This paper describes the design and implementation based on the WS-Trust and WS-SecureConversation specifications supporting an authentication method based on a password, i.e. shared secret. It is the first effort to bring password-based authentication key exchange methods into the message level security, right after Steiner et al. [12] and Taylor et al. [13] ported these methods into the Transport Layer Security (TLS) protocol.

A password is a short string chosen from a relatively small dictionary so that it is easier to be memorized than a long symmetric key; however, passwords are subject to various attacks such as dictionary attack and network eavesdropping. Therefore, passwords should not be used directly as input of signature/encryption schemes. A run-time password-derived secret should be used instead.

Our protocol implementation consists of a password-based authenticated Diffie-Hellman key exchange to agree on a session key, and a key derivation to educe multiple session keys from this master key. Each session is in turn used in conjunction with a symmetric cipher such as the AES, and a Message Authentication Code such as the HMAC, to implement secure message exchanges. The communications for passing these inner cryptographic primitives and parameters need to be defined for WSRF-compliant clients and services. This requires the definition of operations for all client/service interactions in Web Service Definition Language (WSDL). WSDL is the standard language in XML for defining Web Services interfaces.

Currently, most of the message level security solutions in Web Services are based on Public Key Infrastructure (PKI), for example, the GSI-SecureConversation implementation of Globus Toolkit 3 and the WS-SecureConversation implementation in the coming release version of WSS4J [17].

The rest of this paper is organized as follows. In Section 2, we illustrate the security context establishment using the WS-Trust and WS-SecureConversation specifications. In Section 3, we introduce the password-based key exchange method and explain its integration in WS-Trust and WS-SecureConversation under the Web Services Resource Framework (WSRF). Section 4 concludes this paper and presents our future directions.

2. WS-TRUST AND WS-SECURE-CONVERSATION IN SECURITY CONTEXT ESTABLISHMENT

2.1 WS-Trust

The Web Services Trust Language is an extension to WS-Security. It defines syntax for security token exchanges to build up trust relationship among different web service domains. It also provides a set of mechanisms to allow a range of security protocols to fit in, such as the Web Services Trust Model. In this Web Services Trust model, if the service requestor does not have the required tokens for a target service, it turns to an authority for them. Such an authority is called a Security Token Service. WS-Trust also defines multi-message exchange mechanisms, such as the challenge-response protocol. Most of the tokens used in WS-SecureConversation are actually defined in WS-Trust, including the most frequently used RequestSecurityToken (RST) and RequestSecurityTokenResponse (RSTR).

2.2 WS-SecureConversation

The Web Services Secure Conversation Language is based on WS-Security and WS-Trust to allow security context establishment, sharing, and session key derivation. It defines a Security Context Token (SCT) shared among the communicating parties for the session lifetime. A SCT usually contains an identifier pointing to the security tokens being shared.

WS-SecureConversation gives three scenarios for establishing security contexts. The first one is SCT created by a security token service; the second one is by one of the communicating parties

and propagated with a message; the last is through negotiation and exchanges. These scenarios, however, do not exclude each other. For instance, a security token service may have to create security context tokens through negotiation.

Our implementation is based on an unreleased version of WSS4J, which includes a preliminary implementation of WS-SecureConversation through a pair of handlers. The interaction between the different components is shown in Figure 1, which assumes that both sides have previously exchanged their public keys. The workflow is as follows:

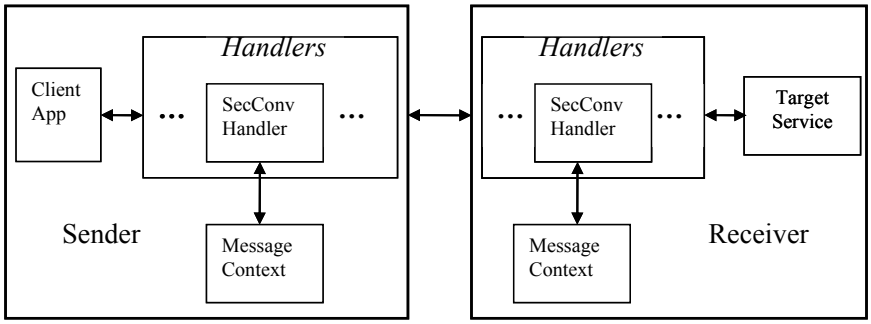

Figure 1 WS-SecureConversation in Handlers

1. The sender, which is usually a client, defines the cryptographic algorithms to use, and generates a random session key suitable for the selected algorithms for either signature or encryption or both.
2. It then encrypts the generated session key with the public key of the receiver (with the assumption that the sender already knows where to get the receiver's public key).
3. It embeds the encrypted session key into a SOAP message. If necessary, he may choose to encrypt the message with the session key.
4. The receiver, upon receiving the SOAP message, decrypts the session key with his own private key. From then on, both sides have possessed the session key for further signature and verification or encryption and decryption operations.

The goal of WS-SecureConversation initialization process is to finish the key exchanges before passing the first remote invocation message to the service provider. Intuitively, this solution looks straightforward and easy to be implemented; however, there is a problem with the dependency on the two handlers, as it is unable to engage in multi-round key exchange interactions. The reason lies in the service side WS-Secure-Conversation handler. WSS4J, as a sub-project of Axis, follows the design of Axis. Axis is an open source SOAP engine for building generic Web Service applications [16]. Simply speaking, every SOAP message goes through the Axis SOAP engine to its desired target along a pre-configured path. The path is made up of a chain of processor nodes, also known as handlers. For example, the encryption handler is responsible for encrypting or decrypting SOAP messages when it receives them. The service provider handler is a special handler that contains the real service logic and sits at the end of the path. Except the service provider, a handler can never short-circuit the message flow by returning the message without passing it on to the next handler on the path.

In a secure Web Service, the WS-SecureConversation handler at the service side is not the service provider but one of the underlying serving handlers. It must faithfully pass the remote invocation message on, until it reaches the service provider, even though the key exchange interaction may not have finished and the session key may not be ready. Therefore it is impossible for a server-side WS-SecureConversation handler to finish a multi-round interaction. This solution only works for those cases that no further interaction is necessary such as the public key-based one assumed in WSS4J.

Even though WS-Trust and WS-SecureConversation provide some methods to compute a new key, they do not restrict the use of alternative key exchange methods to obtain session keys. These alternative methods include Diffie-Hellman key exchange, independent security token services and other services that propagate the tokens. In order to address the before mentioned restrictions imposed by the WSS4J/Axis implementation, our WS-SecureConversation implementation inherits the approach that GSI-SecureConversation adopted by implementing the negotiation mechanism as a separate service. The details will be addressed in the next section.

3. PASSWORD-AUTHENTICATED KEY EXCHANGE IN WS-TRUST AND WS-SECURECONVERSATION

3.1 A Method for Password-Based Key Exchange

Methods for Authenticated Key Exchange (AKE) allow two parties to agree on a common secret value. This secret value, often refers to as a session key, is stored in a security context. In practice, multiple keys will be derived from this session key to implement mechanisms for message confidentiality and integrity. AKE methods based on passwords uses a (short) shared password as a means to compute this session key. Bellovin and Merritt first raised this problem in 1992 [3] and since then several methods have been proposed to solve it. The AuthA protocol is an example of such a provably-secure method [2, 4, 5].

The AuthA protocol is based on the Diffie-Hellman key-agreement protocol [6]. It offers protection against man-in-the-middle attacks by the flows of the Diffie-Hellman exchange under

the password, provides semantic security of the session key, mutual authentication, as well as forward secrecy. A typical workflow of the AuthA method is depicted in Figure 2. The Client and the Server share a previously known password *passwd*. G is a cyclic group on which the Diffie-Hellman problem is hard. The

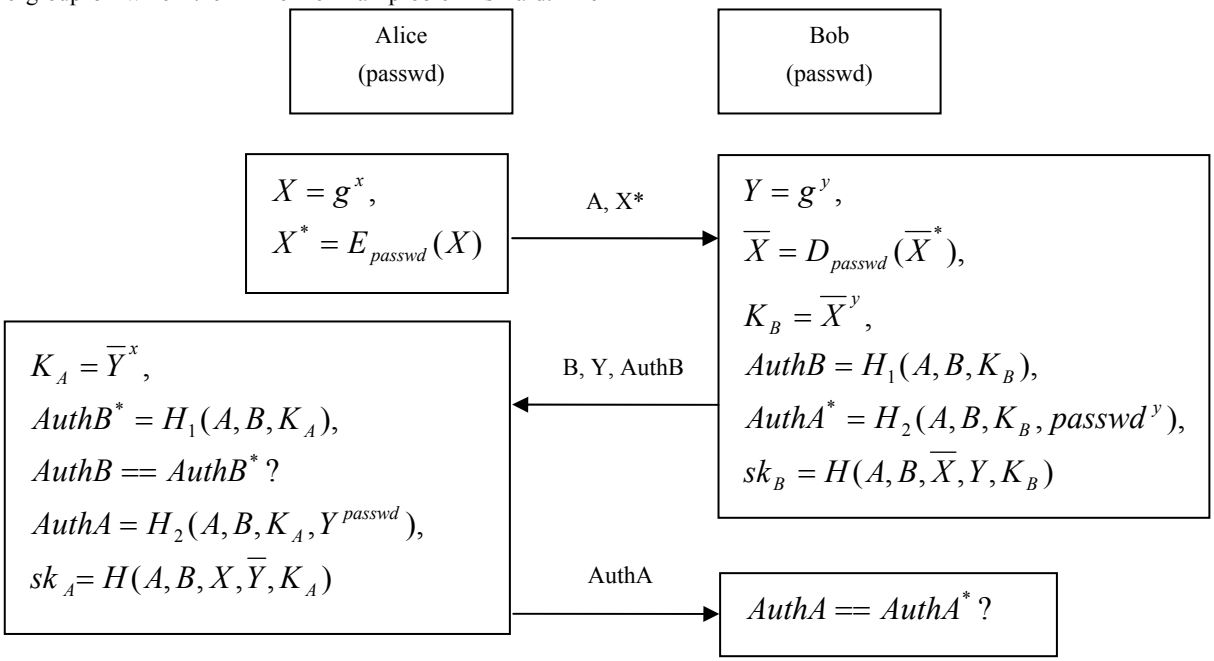

Figure 2 The AuthA Method

group is produced by a generator *g*. By default, *G*, *p*, and *g*, are all well-known.

The Client picks a random value for *x* and calculates *X* from the generator *g*. Instead of presenting *X* to the server as in the conventional Diffie-Hellman protocol, the Client encrypts X with the password *passwd*, and sends the encrypted *X** over to the Server along with its name. Upon receiving this request, the Server chooses at random a value *y*, and calculates *Y* using the generator *g*. Meanwhile, it decrypts *X* with the password *passwd* from *X**. Considering that *X** could have been modified or replaced by a third-party attacker, we mark the received *X** as $\overline{X}$*, and the decrypted *X* as $\overline{X}$. If no attacker has modified these values on the wire, these values are as follows $X^* = \overline{X}^*$ and $X = \overline{X}$, . Upon receiving *X*, the Server computes the Diffie-Hellman secret value key K_B. It also then computes the session key *sk* using the Diffie-Hellman secret value and some additional information including the password and the names of both parties.

The AuthA protocol achieves the property of mutual authentication by having the Client and the Server compute and send the authenticators *AuthB* and *AuthA**. For efficiency purposes, the Server sends the values *Y*, *AuthB (*as well as his name), in one single message to the Client rather than two separated messages. After receiving *Y*, the Client computes the session key and then checks whether the received value *AuthB* matches its computed *AuthB**. If so, and the Client computes the session key and *AuthA*, respectively. If not, the Server cannot be authenticated, and the key-exchange fails. Lastly, the Client returns to the Server *AuthA* to authenticate itself. If *AuthA* matches *AuthA**, the key-exchange terminated successfully and the two parties share the same value for the session key *sk.*

In Figure 2, the session key sk_A is equal to sk_B, as long as $X = \overline{X}$, and $Y = \overline{Y}$; both parties will obtain the session key without exchanging it explicitly over the wire.

3.2 WS-SecureConversation Design Criteria

As we explained in section 2.2, because the WSS4J solution does not support multi-round interaction cases such as the AuthA key exchange method, we have to provide our own solution to support the AuthA and any other interaction requirements.

In our WS-SecureConversation implementation design, we adopt the GSI-SecureConversation approach by implementing the server side negotiation mechanism as a separate service in the same container as the application services. A WS-SecureConversation handler is provided at the client side.

This approach leverages the Web Service Resource Framework (WSRF) by treating the service security context as a stateful resources.

The client side WS-SecureConversation handler interacts with the remote WS-SecureConversation service for as many round trips as needed. The whole workflow is shown in Figure 3, illustrated in two phases. The first phase is the initial interaction for session key exchange between the client side hander and the remote service of WS-SecureConversation; in the second phase, both sides communicate with each other with the derived session key stored in their contexts.

The steps of the first phase are as follows:

1.0. The client application starts an initial remote call through a SOAP message.

1.1. The SOAP message is handled by the WS-Secure-Conversation handler, which finds that a secure conversation context is supposed to be established. It then blocks the message

key will be agreed upon and stored in both sides' session contexts. At service side, the session context lies in the service's resource.

With the security context established, the WS-Secure-

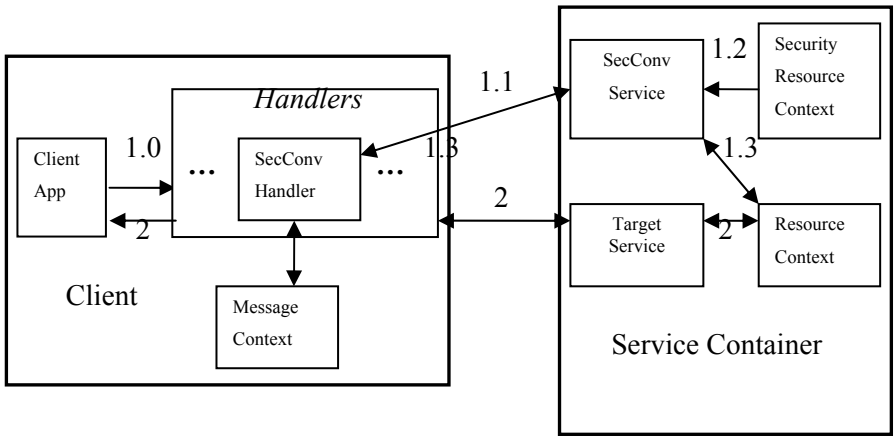

Figure 3 Server side WS-SecureConversation

and makes a remote *RequestSecurityToken* (RST) call defined in WS-Trust to the WS-SecureConversation service, which is located in the same service container as the remote target service. Depending on the key exchange method, the SOAP message may contain the selected methods and algorithms, the required parameters, public keys, entropies, generated random session keys and other information.

1.2. As the WS-SecureConversation service receives the RST incoming message, it interprets the information attached in the

Conversation handler passes the first SOAP message on to the next handler, and thus the client begins to interact with the real target service in phase 2.

The model in Figure 3 could be generalized with an independently functioning Service Token Service, described in WS-Trust as a typical WS-Trust model, as Figure 4 shows. The service plays the role as a security authority. However, an externalized Service Token Service means extra trust relationship to be established between the Service Token Service and the

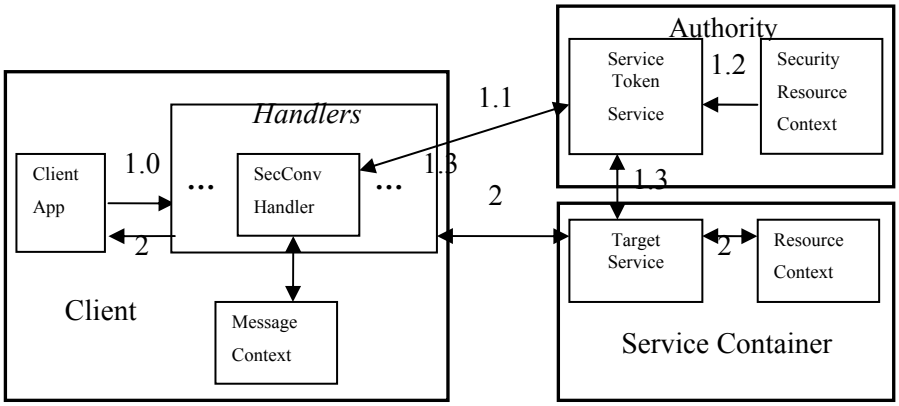

Figure 4 WS-SecureConversation Service Externalized as Security Token Service

SOAP message according to its knowledge to the specified method, which is defined as schemas in most cases. It then fetches the security related information, such as credentials, from a security context resource if necessary. The security context resource has the general information about the owner who is running the service container.

1.3 The initialization interaction may continue for more than one round trip. The following SOAP message exchanges are under the name of *RequestSecurityTokenResponse* (RSTR) call, defined in WS-Trust. The RST with one or more *SecurityContextToken*s (SCT) is returned. A SCT has an identifier referring to a token established or to be established in the contexts. Finally, a session

target service. The service itself needs a full fledged authentication and authorization mechanism. Moreover, the WS-SecureConversation service will no longer interact with the target service's resource internally, but through the interfaces provided by a standard stateful service. Separating the WS-Secure-Conversation service from the targeting service container helps alleviate the load, as we can deploy or move the WS-Secure-Conversation service to another host.

3.3 Integration of Password-based Key Exchange in WS-SecureConversation

Based on the implementations of the AuthA and WS-SecureConversation, the integration work is straightforward. In our WS-SecureConversation implementation, to allow different security context establishment methods to be integrated, the method general behaviors are abstracted as interfaces *ClientNegotiator* and *ServerNegotiator* for both client side handler and server side service, respectively. To integrate the AuthA method, we had to implement the corresponding interfaces called *AuthAClient-Negotiator* and *AuthAServerNegotiator*. The AuthA method specific parameters are grouped as customized tokens, which are defined in XML Schema, including *ClientInitToken* (CIT), *ServerResponseToken*, (SRT) and *ClientResponseToken* (CRT). Each token contains one or more cryptographic parameters, such as the public keys and the authentication bits. These parameters can be either required or optional, depending on the working mode.

The instances of *AuthAClientNegotiator* and *AuthAServer-Negotiator* are responsible for processing those tokens. WS-SecureConversation handler and service wrap up them with the WS-SecureConversation specific tokens, when processing a response. When dealing with a request, they unwrap the WS-SecureConversation specific tokens for protocol specific tokens. The instance of *AuthAServerNegotiator* is stored in the resource context for the current session, so that in the next round, WS-SecureConversation service could fetch it from the resource context according to the session context ID number.

3.4 Status

The AuthA method is implemented in Java in the form of a library, with a dependency on Bouncy Castle library [20]. The WS-Trust and WS-SecureConversation implementations are built on top of the pre-release version of Globus Toolkit 4.

4. CONCLUSION AND FUTURE WORK

In this paper, we brought the password-based authenticated key exchange method to the message level security for run-time session key derivation. We first briefly described the WS-Trust and WS-SecureConversation specifications, which are emerging message level security specifications for security context establishment, sharing and derivation among multiple trust domains. We then introduced the AuthA method as a standardized password-based AKE method. Finally, we illustrated how we have implemented a working system based on those specifications and the WSRF-compliant Globus Toolkit.

Our future work will first be dedicated to the hardening of the implementation such that we can feel confident that our implementation can be deployed in real production-like environments. High on our priority list is also to add One-Time Password (OTP) features [19] to the AuthA protocol using Abdalla et al.'s recent work [1]. We also plan to add mechanisms to the AuthA protocol that would add resistence against Denial of Services (DoS) attacks. The number of Denial of Services (DoS) attacks through the Internet has grown tremendously in the last couple of years. The effectiveness of DoS attacks can be decreased through the use of specific cryptographic mechanisms [4], which treat the amount of Perfect Forward-Secrecy (PFS) as an engineering parameter that can be traded off against resistance to DoS attacks.

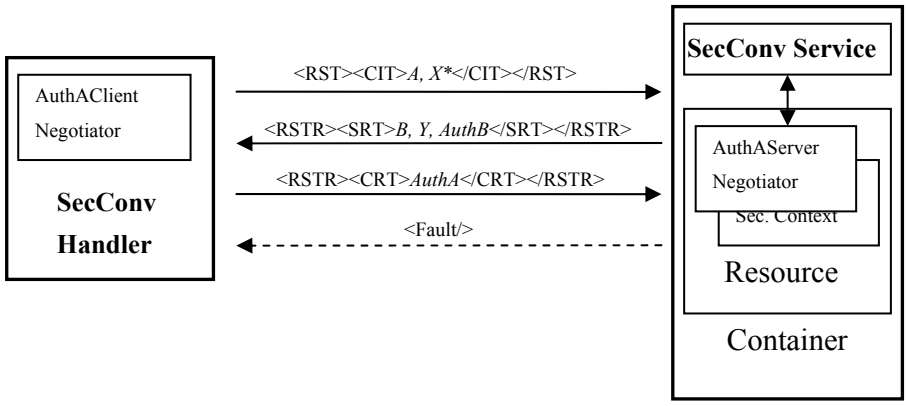

Figure 5 WS-SecureConversation Interactions in AuthA

In addition, we have implemented the signature and encryption handlers. These handlers have two tasks. The first one is to fetch the secrets from the security contexts. The security context is part of the resource context at the service side; while it is incorporated in the background message context at the client side. The security context resources may be stored in a persistent storage such as a database, supported by the WSRF layer. The physical location of a security context is transparent to the handlers. The other task is to sign or encrypt the desired sections of the SOAP envelope with the symmetric keys obtained. The whole interaction process is depicted in Figure 5.

Furthermore, the mutual authentication version of AuthA method takes one and a half round trip. However, in practice, it takes two full round trips, as the client's call to the service has to be returned, even with an empty envelope for the last message. We are investigating whether we could further optimize the use of the protocol by also exchanging protected application messages during the second round-trip, which would effectively reduce the overhead of the authenticated key exchange to a single round trip.

5. ACKNOWLEDGMENTS

We are pleased to acknowledge contributions to the implementation by Rachana Ananthakrishnan and Jarek Gawor,

and we would like to thank Dennis Gannon and Ian Foster for their continuous support. The authors also thank David Pointcheval for his invaluable discussions on cryptographic issues related to this document.

Our co-author Olivier Chevassut, was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, Mathematical Information and Computing Sciences Division, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. This document is report LBNL-56361. Disclaimer available at http://www-library.lbl.gov/disclaimer.

6. REFERENCES

[1] Abdalla M., Chevassut O., and Pointcheval D., Corruption in Password-Authenticated Key Exchange, Submitted for publication, August 2003. Also available as LBNL Report Number LBNL-56212.

[2] Bellare M. and Rogaway P., The AuthA Protocol for Password-based Authenticated Key Exchange. March 14, 2000.

[3] Bellovin S. M. and Merritt M., Encrypted Key Exchange: Password-Based Protocols Secure against Dictionary Attack, *the Proc. Of the Symposium on Security and Privacy, IEEE,* 1992, pp. 72-84.

[4] Bresson E., Chevassut O., and Pointcheval D., New Security Results on Encrypted Key Exchange, *the 7th International Workshop on Theory and Practice in Public Key Cryptography,* March, 2004.

[5] Bresson E., Chevassut O., and Pointcheval D., Security Proofs for an Efficient Password-Based Key Exchange, *the 10th ACM Conference on Computer and Communication Security*, Oct., 2003.

[6] Diffie W. and Hellman M., New directions in cryptography, *IEEE Transactions on Information Theory IT-22*, 6 (Nov.), 1976, pp. 644-654.

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

[8] Foster I., Kesselman C., Globus: A Metacomputing Infrastructure Toolkit, *Intl J. Supercomputer Applications,* 11(2):115-128, 1997.

[9] Foster I. and Kesselman C., Computational Grids. Foster I. and Kesselman C. eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, 248. 2002.

[10] Novotny J., Tuecke S., and Welch V., An Online Credential Repository for the Grid: MyProxy, *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press,* Aug. 2001.

[11] Shirasuna S., Slominski A., Fang L., and Gannon D., Performance Comparison of Security Mechanisms for Grid Services, *the 5th IEEE/ACM International Workshop on Grid Computing,* Pittsburgh, Nov. 8, 2004.

[12] Steiner M., Buhler P., Eirich T., and Waidner M., Secure Password-Based Cipher Suite for TLS, *the Proceedings of Network and Distributed Systems Security Symposium,* San Diego, CA, Feb. 3-4, 2000, pp. 129-142.

[13] Taylor D., Wu T., Mavroyanopoulos N., and Perrin T., Using SRP for TLS Authentication, *TLS Working Group, IETF Internet Draft,* August 2004.

[14] Welch V., Siebenlist F., Foster I., Bresnahan J., Czajkowski K., Gawor J., Kesselman C., Meder S., Pearlman L., Tuecke S., Security for Grid Services, *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press,* June 2003.

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

[16] Apache Web Services Project, Axis, http://ws.apache.org/axis.

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

[18] BEA, Computer Associates, IBM, Layer7, Microsoft, Netegrity, Oblix, OpenNetwork, Ping Identity, Reactivity, RSA Security, VeriSign, Westbridge, Web Services Trust Language, March 2004. "http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-trust.asp"

[19] IETF, A One-Time Password System (RFC 2289), February, 1998. http://www.ietf.org/rfc/rfc2289.txt.

[20] Legion of Bouncy Castle, Bouncy Castle library, http://www.bouncycastle.org.

[21] OASIS, “Web Service Resource Framework”, March 2004.

[22] OASIS, “Web Services Security: SOAP Message Security” March 15 2004.

[23] BEA, Computer Associates, IBM, Layer7, Microsoft, Netegrity, Oblix, OpenNetwork, Ping Identity, Reactivity, RSA Security, VeriSign, Westbridge, “Web Services Secure Conversation Language” May 2004. "http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-secureconversation.asp"

[24] Open Grid Services Architecture, OGSA-working-group at GGF, https://forge.gridforum.org/projects/ogsa-wg.