A Multi-party Implementation of WS-SecureConversation

Hongbin Liu, Geoffrey Fox, Marlon Pierce, Shrideep Pallickara

Community Grids Lab, Indiana University, Bloomington, Indiana 47404

1 Introduction

Grid computing is an emergent computing paradigm focusing on solving the problems resource sharing in heterogeneous environments of science, engineer and commerce [1, 2, 3]. It has made significant progresses in large scale data management and access, resource naming and discovery, information services, as well as building innovative grid services to integrate existing applications. Security has been one of the most important areas in grids. Researchers on security have been isolating typical grids usage scenarios [4], identifying unique security requirements [5], and proposing efficient schemes of authentication and authorization [6, 7]. Whereas transport and network level protocols like SSL [8] and IPSec [9] provides working solutions to client-server model, they are not sufficient to meet more complicated requirements in grids. In its recent movement, grids have adopted Web services technology [1] to deal with environmental heterogeneity and to enhance service and application interoperability. As SOAP 1.2 [10] becomes widely accepted as the XML messaging standard, the limits of traditional solutions are further recognized.

Group communication and its security have been thought essential to grids [5]. They have witnessed an increased interest as the popularity and diversity of collaborative applications continue to grow. Scientific cooperation in grids [2], peer-to-peer online sessions [11], audio-video conferences [12], all of them use, or can benefit from using, group communications. Although the threats to the group communication are similar to those to unicast applications, because of its broad scope, approaches to solving the security problems in the group communication differs [13].

This paper reports on our work in implementing a multi-party capable WS-SecureConversation (WSSC) [14]. WSSC is a specification for securing XML messaging and an extension of W3C standard WS-Security [15]. They both provide integrity and confidentiality protection of a SOAP message through mechanisms that are independent of specific security models and cryptographic algorithms.

The multi-party WSSC is intended to be a security module that follows the XML and Web services standards. It can be used in the group communication middleware infrastructure. Although we focus on NaradaBrokering [16] as such a middleware infrastructure, nothing prevents it from fitting into others as long as they can work with XML messaging.

The rest of the paper is organized as follows. In section two, we briefly compare SSL to XML messaging security and outline our future work towards a secure group communication middleware infrastructure. We devote section three to introducing the multi-party implementation. Performance data of the implementation is presented in section four. We then conduct a vulnerability analysis of the implemented WSSC in section five. Related work is introduced in section six and we conclude with section seven.

2 Design Goals and Future Direction

In this section, we compare the role of SSL to that of WS-Security and WSSC in XML messaging security, lay out the design goals of our WSSC implementation and then describe future work related to it. In Fig. 1, there is zero or more intermediaries or relay nodes between initial SOAP sender and ultimate receiver for a specific message. Depending on the "role" in the header attributes, SOAP allows the relay nodes to access and modify the message content. In the past when there is no message security available, the neighboring nodes are typically linked by individual SSL sockets; and it's argued that aggregately they can realize protection in the overall system. XML message level mechanism represented by WS-Security and WSSC address the following shortcomings of SSL like solution in SOAP relay.

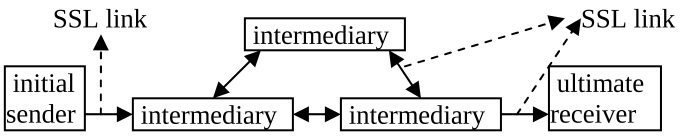

Fig. 1 SOAP message relay

1) Between the initial sender and the ultimate receiver a sequence of SSL connections must be established.
2) Intermediaries must do individual link encryption/decryption. If a single link is compromised, all the messages passing the link are compromised.
3) Security protection is done with the channel; it's not possible to do partial or chosen message protection in a given channel.
4) All SSL connections are desired to keep persistent over application sessions, a shut-down and recovery of the intermediaries don't automatically recover the on-going communication between the initial sender and the ultimate receiver.

Based on this analysis, we think transport solutions such as SSL and VPN do not stand to be competitive in an environment where messages are routed in open public networks and authorized processing and partial modification can happen during transmission. WS-Security and WSSC protect a message instead of the transport link. A WSSC implementation can replace individual SSL links in Fig. 1 and supports secure conversation among multiple participants in a group setting. This brings up the fifth limitation of SSL being used in grids.

5) Secure group communication is realized by multiple secure unicast channels.

The design goals of the WSSC implementation are as follows.

- Fully compatible with the WSSC specification [14] and work seamlessly with SOAP relay requirements including partial and chosen message protection and intermediary intervention.
- Provide message level privacy and integrity to group communication, short of dynamic rekeying [17] support.
- Mitigate replay attack and attempt at decreasing the likelihood of DoS attack.

Our WSSC implementation is one step towards our ultimate goal of providing secure and highly efficient messaging middleware to grids community. To achieve the goal, two requirements must be satisfied. The infrastructure [16] must be secured against various attacks that can be launched in open network; and flexible and tunable access controls are available to communication ends including publishers and subscribers. We propose the rest milestones with the goal and requirements in mind.

- Devise an efficient rekeying scheme and map the rekeying data structure to the messaging middleware NaradaBrokering. Such a mapping is desired because the existing broker network may not naturally be in congruent with update-tree structure used in innovative rekeying algorithms. Further information about rekeying can be found in the later section.
- Design an authorization scheme for publish subscribe paradigm. The work will unify authentication and access control, and result in programmable trust management [30] for brokers and for publishers and subscribers of a certain topic in a dynamic network condition. Part of the on-going work is in [18].

We will be able to report our progress with respect to the two tasks in near future.

3 Multi-party Implementation

3.1 Overview of relationship of WS-Security and WSSC

As a framework for messaging security, WS-Security provides XML vocabulary that is to be used to build a secure message untied with specific cryptographic algorithms. For example, TripleDES and AES, or RSA and HMAC are all equally possible to be asked from calling applications. It is also extensible in that new authorization mechanisms can be just added to be supported as X.509 digital certificate currently is.

WSSC describes ways to build a security context in a communication session, to achieve key establishment and to employ the key in the context of the session. It is one of WSSC's design goals [14, section one] to take techniques such as message sequencing and time-stamping to cope with possible threats that are left unaddressed purposedly in WS-Security [15, section 7]. Replay attack [19] is such an example. Applications that have multiple-message exchange should use WSSC instead of plain WS-Security or come up with their own modules to deal with the attacks. Additionally, for some sessions where the number of application messages exceeds a certain threshold, WSSC can result in better efficiency despite the cost incurred for key and security token exchange at the beginning of the session.

Among those XML elements supplied by WS-Security, the following are important and not missing in WSSC. A Signature element should be included in the SOAP header if any part of the message has been performed with digital signing. The element has children elements specifying the following information: a) what part of message is signed? This can be specified via a URI attribute referencing an id of the signed element; b) what is a signing algorithm? c) what kind of transformations, including message digestion and canonicalization are performed before signing actually takes place and what algorithms are used for the transformations? d) what is the key used? This can be a private key or a shared secret, resolved usually via a KeyInfo element, which can contain an EncryptedKey or a reference to binary token (encoded in Base64) in the header or other kind of security token supposedly known to recipient; e) finally, a signature value. If any part of the message is encrypted, it should be replaced with an EncryptedData element, which provides the following information: a) what part of message has been replaced (or encrypted)? b) what is the encryption algorithm used? c) what is the key used? Again, information should be provided leading to a key resolution; d) finally, a ciphertext.

3.2 Other XML constructs and objects in WSSC

We now introduce informational elements -- three XML constructs and two objects that are functionally vital in WSSC.

SecurityContextToken contains UUID, which serves as an identification of a shared secret. The secret and UUID should be kept in memory by session participants. The token can have an id attribute which can be used for reference within the SOAP message.

RequestedProofToken contains encrypted key material used as the shared secret. RequestedProofToken and SecurityContextToken co-exist side by side and are sent to whoever is supposed to expect a shared secret. The generation and dispatch of the shared secret can be made by one of communication ends, a trusted security token service, or some kind of negotiation process between the parties.

DerivedKeyToken contains several pieces of information that serves as input parameters for key derivation algorithm. The algorithm computes a derived key. It optionally contains a reference which points, within the message, to a SecurityContextToken.

SequenceNumber and Timestamp The implementation provides an API to specify TTL (Time-To-Live) in a Timestamp. Each conversation session is associated with a monotonically increasing sequence number, which is initialized as a random integer. Both SequenceNumber and Timestamp are protected from unauthorized tampering.

Security context. The principal object in a security context is the shared secret, which will be used in cryptographic operations including key derivation. There are three methods of establishing a security context in WSSC, but involved trust relationship between context provider and user is not specified. Security context has one-to-one mapping relationship with SecurityContextToken.

Conversation session. WSSC doesn’t define what the session is. We think multiple security contexts are allowed to simultaneously exist in a session, for example, one for signing and one for encryption. Another example is that applications like to use different keys in different modules which are desired to share a session. Nor does it clarify the relationship between the session and the security context. A secure conversation session has an id, which is shared among all session participants. The session holds one or more security contexts; each of them is associated with a shared secret. Initialization of a new session is done in tandem with the security context establishment. Once the new session is ushered in, more security contexts can be added. SecurityContextToken functions as an identification mechanism to the shared secret, which must be stored and can be accessed during the entire session. To further clarify what the session is, we would like to propose the following observations.

- A session contains one or more security contexts, but the establishment of security context is independent of the session creation.
- When, how and under what circumstances to end the session is within the implementation domain. Ending the session may or may not be tied with the immediate removal of security context.
- Shared secret can be disassociated with session and/or destroyed completely during or after the session.

Group communication security can be realized by employing a group key [19]. This key should be changed on every membership change, which poses a scalability challenge for large dynamic groups. The rekeying in a dynamic group is complicated and should be treated with care. So far, we have focused on how secure messaging can be achieved once a group key is distributed. This group key will be utilized for both privacy and integrity protection. We implement a simple key distribution scheme in compliance with WS-Trust [20], using the assumption that a central group controller has access to the digital certificate of each participant.

3.3 Feasibility of multi-party implementation

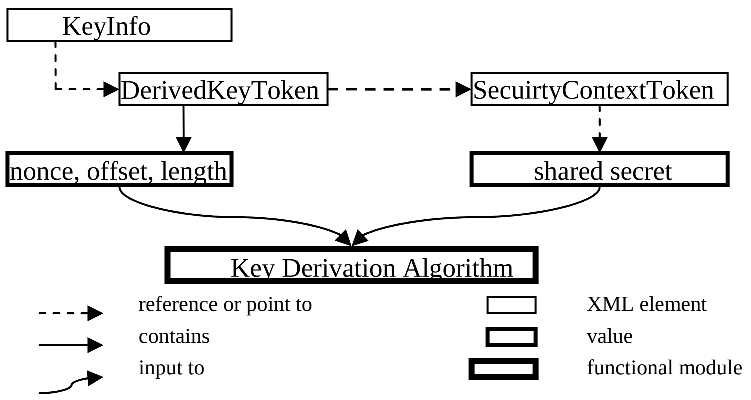

Fig. 2 Reference chain leading to key derivation

Establishment and distribution of a shared secret key among a group of participants in a secure and efficient way is a formidable task. WSSC is not designed to be machinery for secure group communication, so it doesn't invent vocabulary or framework for group key establishment. Particularly, the group key rekeying problem [17, 24] is not addressed. It indeed describes three guidelines on how a secret can be distributed, suitably used in two-party conversation. We provide generalized implementation of key distribution (key issuing in WS-Trust) and adopt them with multiple participants. We will attempt at the rekeying problem set in the context of the middleware infrastructure in our next milestone.

Once the shared secret is established, the participants can then rely on the secret for message protection. Using a symmetric encryption key many times, however, can result in weakness before chosen plaintext attack. Key derivation is recommended in WSSC as a defensive mechanism and we take it seriously. Input information (nonce, length and offset) to the derivation is inserted into XML by encryption side and passed to decryption side, who then independently derives the encryption key. In Fig.2, KeyInfo element existing in Signature/SignedInfo or EncryptedData contains a SecurityTokenReference element, which points to DerivedKeyToken in the message. This token provides some necessary values as input to the key derivation algorithm. It also contains SecurityTokenReference to SecurityContextToken, which identifies the rest of necessary values – shared secret, to the key derivation. The token can also set the algorithm used, though currently only Psha1 is supported in the implementation. The steps are independent of how many participants are in the session. As long as DerivedKeyToken is received and the shared secret is there, key derivation will be done. Furthermore, the steps apply both to sender/assembly side and receiver/disassembly side. In other words, after a key is derived, it can be used either in encryption/signing, or decryption/signature verification.

Since cryptography in WSSC is based on the shared key, we employ the technique of Message Authentication Code (MAC) [19] is symmetric key signing. Currently, HMAC/SHA1 [21] is the only algorithm available in our implementation for calling applications.

3.4 Implementation details

3.4.1 data structures

- *ConversationSession* has fields of sessionId, beginTime, startSequence, SecurityOptions, a table of StoredSCTs and a table of StoredDKTs.
- *SecurityOptions* is a conglomeration of cryptography operation related parameters which are input from WSSC calling parties, for example, where the digital certificate is, what algorithm to use for data encryption, what nonce value to use for next key derivation. Currently there are 28 such parameters.
- *StoredSCT* is in-memory structure counterpart of SecurityContextToken. It has an id and a byte array for a shared secret.
- *StoredDKT* is in-memory structure counterpart of DerivedKeyToken. It has all the fields that are defined for the token in WSSC.
- *SessionTank* is a table of ConversationSessions. This is written as a book-keeping class, which keeps track of currently active conversation sessions.

3.4.2 session management

The conversation session management is illustrated in the state-machine diagram of Fig. 3. The dashed arrow represents a type of state transition where no information is sent back to the last relay node when some processing error is thrown. The details of the error will be reported to the logging facility of calling application. The second type transition is transition response, used in a few cases where it is appropriate to send error information back over the network. Choice between transition silent and transition response is made depending on what kind of error has occurred and the possibility of denial of service attack associated with the error. Block error in Fig. 2 represents normal transition of state.

The first question is when and how a new conversation is started. Unfortunately, WSSC doesn't clearly specify the mechanism to initiate a session. In a similar session orientated protocol SSL, we see a handshake protocol is used to get a new session initiated. At the end of the handshake, two important data structures among those that must have been established and shared between client and server, one is a session id; the other is cipher suite which includes shared secrets used for encryption. A carefully designed handshake protocol is critical for preventing and detecting attacks, besides obtaining necessary, functional data to conduct the rest of

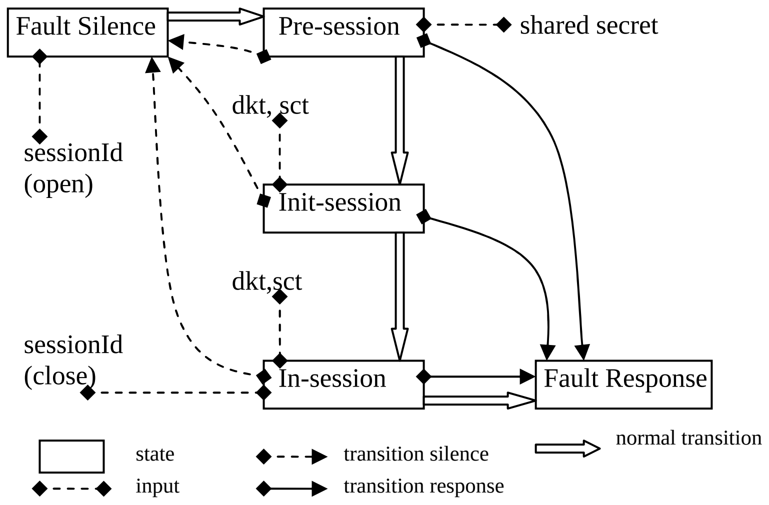

Fig. 3 Session state transition diagram

the session. Considering the lack of sufficient specification on session initiation and considering the status of WSSC, we decide not to implement a handshake protocol (at least currently). Instead, we envision that application will do what is necessary for the session initiation. To aid to application level session management, the implementation provides the following calling interfaces.

- *startSession()* creates a new conversation session, initializes necessary data structures and return a new session id.
- *endSession(sessionId)* does safe shut-down operations, especially removing sensitive information such as shared secrets and DerivedKeyToken parameters.

- *getSCT(sessionId,,securityContextId)*, *addSCT(sessionId,SecurityContext)*, *removeSCT(sessionId,securityContextId)* gets, adds, or removes SecurityContextToken from the specified session, respectively.
- *checkSeqNumber(sequenceNumber)* checks if the input sequence number is "safe" or not.
- *setBeginTime(beginTime), getBeginTime()* set or return the session start time.
- *setSecurityOptions(sessionId, securityOptions)* and *getSecurityOptions(sessionId)* set or get security parameters for the session.

Before a session begins, the session id is transmitted from either a trusted group controller or a group peer. If the id is accepted, the new session gets started and its beginTime is set. Typically, one or more shared secrets identified by SecurityContextToken and embedded in RequestedProofToken will be sent along with the new session id. There are two known scenarios, however, where the id can be transmitted alone. If some security contexts have been previously established and is intended to be used in this new session, then the shared secret itself doesn't have to be transmitted again. In this case a receiver expects a SecurityContextToken in order to copy the secret into the session. In another case, the secret can be transmitted later. It's not likely but possible that the receiver will never see the incoming secret associated with the id. To prevent empty session (without any shared secret and never be used) from accumulating, a background thread periodically wakes up and checks to remove those empty ids. But this preventive feature is not currently implemented.

4 Performance Results

Soap size	enc	dec	sign	verify	e/s	p-e/s	s/e	p-s/e
1k	4	16	8	8	10	23	11	25
2k	7	14	5	9	13	23	13	27
10k	23	30	13	10	30	41	29	42
100k	402	307	48	27	420	327	433	329
1000k	2000	1697	340	165	2157	1869	2260	1929

Fig. 4 WSSC performance results (in millisecond) with varying SOAP sizes

We collect timing data over local operations, that is, no network transmission has happened to input documents. By neutralizing network condition, we are able to focus on relatively more important performance issues [29]. It also has an advantage of simplifying the multi-party measurement – cost of secure conversation is modeled as summation of transmission time plus local operation time. We view that the latter is dependent upon several factors: size and complexity of SOAP on which cryptographic and XML processing are performed, choice of cryptographic algorithms, and of course, efficiency in implementation. We present a segment of our results (due to space limit) in Fig. 4, which is measured over five SOAPs, ranging from 1 kilobyte to 1 megabyte. In the figure enc is short for encryption and dec for decryption. e/s means encrypt and signing are performed, s/e also covers them both but in reverse order. p-e/s column contains times of processing documents that are both encrypted and signed.

The experiments are conducted on Pentium 4 box with Linux kernel version 2.4.10, Hotspot Client JVM with JRE version J2sdk1.4.2 and JCE provider BouncyCastle. The result set is verified at another Pentium 4 Windows XP to avoid possible gross errors. Each data entry is obtained as 64 bit integer averaged over 100 runs.

5 Security Analysis

A message can come under a reply attack [19] even if the payload is signed and encrypted. The common defensive mechanisms against replay attack include using sequence number, timestamp, or some correlation techniques. In our implementation, a sequence number is initially randomly generated and will monotonically increase. If an incoming sequence number is less than the current one at receiver, the silent fault will be immediately thrown, logged and reported to the upper level layer.

The proof of possession of the private key is necessary when a digital signature is used for identity checking. A specific example of the lack of sound identity checking is as follows. In WS-Security, in the very first contact, sender embeds her digital certificate as a BinarySecurityToken inside the message. She can generate a symmetric key and encrypt the key with her private key and send the encrypted key over. Receiver will retrieve sender's public key from the token and do the key decryption. From then on, the two can communicate securely based on the shared key. Since issuer serial or subject or alias is public data in X.509 digital certificate, a malicious attacker can grab any one of them and just use it. Although she won't be able to decrypt data sent from the receiver in their communication, talking to the devil even it's a deaf devil should be avoided. In the above case, the receiver can use a challenge (a large encrypted number) to address the problem.

Possible DoS attacks [19] for authentication service in WSSC are also addressed. In this case, the attacker uses the collected digital certificates and sends a target a large amount of messages. Public key resolving and certificate chain process in challenge-response consume a considerable amount of resources. An alleviating technique is for the receiver to keep the records of failed authentication and identity checking in the recent past. Then a policy can be made stop processing the message once it is found coming from such a failed source. But the attack can put "source" such as SOAP actor or role in Signature so that to get "source", it's necessary to conduct cryptographic operations. Currently we don't have a solution to completely avoid the problem.

Several units in the security header, including sequence number, timestamp, DerivedKeyToken, and SecurityContextToken, should be signed for integrity protection and so to mitigate replay attack. Signing them unfortunately increases the complexity of header processing at receiver, which can lead to DoS attack. Currently we are considering mechanisms to decrease the complexity without causing compromise to integrity protection.

Due to its relaying nature, SOAP can contain elements authored by different entities. This raises the level of security processing difficulty; because once an error is encountered, it must be decided either to quit processing entirely or to continue somehow. If quit entirely, then it's possible some good elements remain un-processed; but if continue, it can be helpful for DoS attack. In the implementation, once the error is found, the processing engine will mark all elements associated with the actor regarding which the error is raised and avoid processing these elements. But this measure will only be effective if attackers don't spoof SOAP actor attribute.

6 Related Work

Group communication is faced with two basic security requirements [22, 27]: a) inter-group communication confidentiality, b) source authentication. Communication confidentiality is realized with data encryption using a group key. This key should be changed on every membership change, which can pose a formidable challenge for large dynamic groups. The group key can also be used for source authentication. In this case, messages signed using the group key can be verified as "coming from a group member" instead of others. If authentication with particular source is desired, individual secret keys, either private public key pair, or other keying mechanisms must be employed. Most practical solutions to the secure group communication devise a central group controller and focus on the communication efficiency improvement of group key update. The idea is to associate each member with some auxiliary keys to facilitate re-keying, so to decrease the number of update messages the controller sends to members upon membership change. One-way function tree scheme (OFT) [24] is conceived with intention to lessen the severity of controller centralization in Wong-lam scheme [19] to some extent. Each node in the OFT binary hierarchy holds two keys, a node key and a blinded key. The difference between Wong-lam and OFT is that in OFT, besides a node key for every node from leaf up to the root, a member holds a blinded key for every sibling of the node on the path. A one-way function is used to derive blinded key from the node key at the same node. A mixing function is used to derive a node's key from the blinded keys of the two children nodes. Upon membership change, three things happen in OFT: a) the directly affected member, i.e. the newly joining or the sibling of the leaving, is sent by the controller in unicast a new node key; b) it may also be notified the blind keys of its ancestral siblings and then can compute the node keys of its ancestors; c) the controller also does the necessary computation of the blind keys and then multicast them, each encrypted with correspondent node key, to a subtree. Ku and Chen [28] analyze a possible collusion attack on the early OFT scheme and propose a fix.

In source authentication, the efficiency problem occurs particularly with packet stream signing, where signing rate must beat the packet generation rate if it is real-time application. With packet chaining approach [23], only the very first packet is signed, and each packet is appended with the digest of the immediate latter one. Verification side stores the signature in the first packet and then eventually verifies them all at the end of stream. Another interesting method uses asymmetric message authentication code (MAC) [27] to provide tradeoff between efficiency and security.

A sender has a set of keys and it appends a message with MACs, each generated with a key in the set. Each receiver knows a unique subset of the sender keys. Verification succeeds only if every key in the subset finds and matches its correspondent MAC in the message. It may suffer the receivers' collusion problem, in which no single receiver knows the entire sender set but several of them can union in collusion to get the sender key set.

Secure Socket Layer (SSL) is a protocol sitting between application and transport layer in OSI model and provides protection over client-server communication. In SSL handshake, communication ends do either one-way or mutual authentication, negotiate a common acceptable cipher suite and then come up with some shared secret, which will used for data encryption after handshake is done. SSL and WSSC both are session-based, dependent on the shared secret, and flexible with employed cryptographic algorithms. However, SSL doesn't provide end-to-end security; it must encrypt the entire application payload whereas with XML security partial encryption is one of fundamental advantages; finally, SSL is only two-party protocol whereas in this paper we demonstrate WSSC has the multi-party capability.

7 Conclusion

In this paper, we present WS-SecureConversation implementation as multi-party capable solution for secure group communication. We believe the work is the first open work on multi-party WS-SecureConversation and the first to explicitly address secure group communication in XML message environment. It can work seamlessly in and together with wide-range environments where communication security is expected.

8 References

[1] Foster, I., and Kesselman, C., The Grid 2: Blueprint for a New Computing Infrastructure, Morgan Kaufmann; 2nd edition, Nov. 2003.

[2] Berman, F., Anthony, H., and Fox, G., Grid Computing: Making the Global Infrastructure a Reality, Wiley, ISBN 0470853190.

[3] Foster, I., Kesselman, C., and Tuecke, S., the Anatomy of the Grid: Enabling Scalable Virtual Organizations, International Journal of Supercomputer Applications, 15(3), 2001.

[4] Humphrey, M., and Thompson, M., Security Implications of Typical Grid Computing Usage Scenarios, Security Working Group GRIP forum draft, Oct. 2000.

[5] Foster, I., Kesselman, C., and Tuecke, S., A Security Architecture for Computational Grids, in Proceedings of 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.

[6] Butt, A.R., Adabala, S., Kapadia, N.H., Figueiredo, R., and Fortes, J.A.B., Fine-grain Access Control for Securing Shared Resources in Computational Grids, Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM , 2002, Page(s): 206 -213.

[7] Butler, R., Welch, V., Engert, D., Foster, I., Tuecke, S., Volmer, J., and Kesselman, C., A National-scale Authentication Infrastructure, Computer, Volume: 33 Issue: 12 , Dec. 2000, Page(s): 60 -66.

[8] Freier, A., Karlton, P., and Kocher, P., Secure Socket Layer 3.0, http://wp.netscape.com/eng/ssl3/

[9] Kent, S., and Atkinson, R., Security Architecture for Internet Protocol, RFC 2401, Nov. 1998.

[10] Gudgin, M., et al., SOAP Version 1.2 Part 1: Messaging Framework, W3C Recommendation, June 2003, http://www.w3.org/TR/soap12-part1.

[11] Kubiatowicz, J., Bindel, D., Chen, Y., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, G., and Zhao, B., OceanStore: An Architecture for Global-scale Persistent Storage, in Proceedings of ACM ASPLOS, Nov. 2000.

[12] Wu, W., Fox, G., Bulut, H., Uyar, A., and Altay, H., Design and Implementation of Collaboration Web-Service System, Neural, Parallel and Scientific Computations, vol. 12, pp. 391-406, 2004.

[13] Tanenbaum, A., and van Steen, M., Distributed Systems: Principles and Paradigms, Prentice Hall, 1st edition, Jan. 2002.

[14] Anderson, S., et al., Web Services Secure Conversation Language (WS-SecureConversation), May 2004, http://msdn.miccrosoft.com/library.

[15] Nadalin, A., et al., Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), OASIS Standard 200401, http://www.oasis-open.org.

[16] NaradaBrokering project, Community Grids Lab, Indiana University, Bloomington, IN, http://www.naradabrokering.org.

[17] Wong, C. K., Gouda, M., and Lam, S. S., Secure Group Communication Using Key Graphs, ACM SIGCOMM, 1998.

[18] Yan, Y., Huang, Y., Fox, G., Kaplan, A., Pallickara, S., Pierce, M., and Topcu, A., Implementing a Prototype of the Security Framework for Distributed Brokering Systems, in Proceedings of 2003 International Conferences on Security and Management, June 2003, Las Vegas, NV.

[19] Bruce Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C, Wiley 2nd edition, Oct. 1995.

[20] Anderson, S., et al, Web Services Trust Language (WS-Trust), May, 2004.

[21] Krawczyk, H., Bellare, and M., Carnetti, R., HMAC: Keyed Hashing for Message Authentication, RFC 2104, http://www.faqs.org/rfcs/rfc2104.html.

[22] Moyer, M., Rao, J., and Rohatgi, P., A Survey of Security Issues in Multicast Communications, IEEE Network, vol. 13:12-33, Nov.-Dec., 1999.

[23] Gennaro, R. and Rohatgi, P., How to Sign Digital Streams, Lecture Notes in Computer Science (LNCS), vol. 1294, Springer-Verlag, 1997.

[24] Sherman, A.T. and McGrew, D. A., Key Establishment in Large Dynamic Groups Using One-way Function Trees, IEEE Transactions on Software Engineering, vol. 29, NO. 5, May 2003, pp. 444-458.

[25] Wong, C and Lam, S., Digital Signatures for Flows and Multicasts, IEEE/ACM Transaction on Networking, vol. 7, 1999.

[26] Golle, P. and Modadugu, N., Authenticating Streamed Data in the Presence of Random Packet Loss, Network and Distributed System Security Symposium, 2001.

[27] Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., and Pinkas, B., Multicast Security: A Taxonomy and Efficient Constructions, IEEE INFOCOM, vol. 2, Mar., 1999.

[28] Wei-Chi Ku and Shuai-Min Chen, An Improved Key Management Scheme for Large Dynamic Groups Using One-way Function Trees, in Proceedings of International Conference on Parallel Processing Workshops, Oct. 2003, pp. 391 - 396.

[29] Liu, H., Pallickara, S., and Fox, G., Performance of Web Services Security, in Proceedings of 13th Annual Mardi Gras Conference, Feb. 2005.
[30] Blaze, M., Feigenbaum, J., and Lacy, J., Decentralized Trust Management, in Proceedings of the 1996 IEEE Symposium on Security and Privacy, 1996.
[31] Box, D., et al., Simple Object Access Protocol (SOAP) 1.1, W3C Note 8 May 2002, http://www.w3.org/TR/2000/NOTE-SOAP-20000508.
[32] BouncyCastle, http://www.bouncycastle.org.
[33] Apache WSS4J project, http://ws.apache.org/ws-fx/wss4j/.
[34] Fox, G., Pallickara, S., and Parastatidis, S., Towards Flexible SOAP Messaging for SOAP Based Service, in Proceedings of the IEEE/ACM Supercomputing Conference 2004, Pittsburgh, PA.