Performance Comparison of Security Mechanisms for Grid Services

Satoshi Shirasuna Aleksander Slominski Liang Fang Dennis Gannon
Department of Computer Science, Indiana University.
215 Lindley Hall, 150 S Woodlawn Avenue, Bloomington, IN 47405-7104
{sshirasu, aslom, lifang, gannon}@cs.indiana.edu

Abstract

Security is one of the most important features for Grid services. There are several specifications used to add security to Grid services, and some of them have been implemented and are in use. However, since most of the security mechanisms involve slow XML manipulations, adding security to Grid services introduces a big performance penalty. In this paper, we introduce various security mechanisms and compare their features and performance. Our evaluation shows that transport level security (SSL) is faster than message level security, and should be used if there is no special requirement to use message level security. For message level security, WS-SecureConversation is generally fast, but has a scalability problem.

1. Introduction

Security is one of the most important features for Grid services [13]. There are several specifications used to add security to Grid services, and some of them have been implemented and are in use. Since Grid services are generally constructed on top of the SOAP [8] layer, it is natural to add security mechanisms to the SOAP or XML layer in order to make services transport-layer independent. However, security mechanisms implemented in the XML layer are slower than any other transport security mechanism because they involve slow XML manipulations. Because of the performance penalty with current Java implementations, none of these security mechanisms have yet to be widely adopted. Also, because Web service implementations for Grid applications are still relatively new, it is not yet clear to service providers which mechanism is most suitable for their needs. It is necessary to clarify the requirements for secure Grid services, and figure out what mechanism can satisfy those requirements with acceptable performance. As the first step, this paper presents the comparison of different security mechanisms on their performance and features.

Throughout the paper, we use two implementations, Globus Toolkit 3.2 (GT3.2) [12] and XSUL [4], in order to compare the supported functions and the performance. Globus Toolkit is a widely-used toolkit for building Grid applications and services. XSUL (also known as XSOAP4) is a modular Java library that constructs Web services and Grid services. XSUL is being developed at Indiana University.

The rest of the paper is organized as follows. Section 2 introduces several security mechanisms for Grid services and their implementations. Section 3 presents the performance evaluation for different security mechanisms, and Section 4 discusses the result.

2. Taxonomy

This section introduces several security mechanisms for Grid services. This paper focuses mainly on the integrity of messages, so the following discussion does not cover encryption, which concerns the confidentiality of messages. Depending on the layer where security is implemented, there are two categories of security mechanisms, transport level security and message level security. Among the following security mechanisms, only the first one, Server Secure Socket (SSL) [14], is a transport layer security mechanism. The rest of them are message level security mechanisms.

2.1. Server Secure Socket (SSL)

This mechanism uses Server Secure Socket (SSL) with GSI certificates, which are based on X.509 certificates [6]. SSL is lightweight because it does not involve any XML manipulations. Also, except the initial handshake, SSL uses a symmetric cipher, which is known to be much faster than an asymmetric cipher. However, since it is transport layer security, it does not work if the connection includes multiple hops, which is a feature supported by SOAP.

Globus Toolkit 3 (GT3) supports this mechanism as GSI-enabled HTTP-based protocol (httpg). However, httpg is not supported beyond GT3.2 even though it still functions

and users can still use it. XSUL also supports SSL-based security. It is equivalent to httpg.

SSL involves a handshake as its initial phase to establish an encrypted connection. This can be an overhead if each invocation of a service involves this phase. XSUL supports HTTP KeepAlive option to keep the HTTP connection open so that a client only needs to do the handshake once when the client wants to interact with the server multiple times. Unfortunately, GT3.2 does not offer any straightforward APIs to set the HTTP KeepAlive option. A client needs to go through the handshake each time it invokes a service.

2.2. XML-Signature

XML-Signature [10] is a standard for digital signatures for XML documents. The usage of XML-Signature with SOAP messages is described in WS-Security specification [16]. With XML-Signature, each message is signed with X509 certificate (GSI certificate). It ensures the integrity of a message, but it does not support replay-attack prevention. As opposed to WS-SecureConversation, which will be described later, this mechanism is stateless and does not need any initial handshakes. Thus, it is suitable for a single invocation of a service.

GT3.2 supports this mechanism as GSI Secure Message. XSUL also supports the same mechanism.

2.3. WS-SecureConversation

WS-SecureConversation [15] is a relatively new protocol to establish and use secure contexts with SOAP messages. First, a secure context is established between a client and a server. Once the security context is established, following messages are signed using XML-Signature standard. It is faster because it uses a symmetric key to sign messages, but it requires additional round trips to establish a connection. This mechanism is suitable for multiple interactions. Even though this mechanism also uses XML-Signature standard, we use the term, *XML-Signature*, for the stateless mechanism described in Section 2.2, and *WS-SecureConversation* for the stateful mechanism described here.

GT3.2 supports a mechanism called GSI Secure Conversation. GSI Secure Conversation does not follow WS-SecureConversation specification, but its basic mechanism is the same as that of WS-SecureConversation. In this paper we use GSI Secure Conversation to evaluate the performance of WS-SecureConversation. Currently, XSUL does not support this mechanism.

2.4. Capabilities

Unlike the rest of the mechanisms described here, a capability-based system adds authorization information to a message as well as ensuring the integrity of the message. XSUL supports a capability-based authorization method called XCAP. XCAP inserts capability tokens, which describe who can invoke which methods of a service using the Security Assertion Makeup Language (SAML) [11], to a SOAP message. This mechanism allows fine-grained authorization without the user needing a local host account, or adding complicated authorization mechanisms to the application logic.

3. Performance evaluation

3.1. Evaluation environment

We measured the performance of the security mechanisms described in Section 2. First, we measured the performance of a simple service, echo service, to evaluate the overhead caused by adding security to a service (Section 3.2). Secondly, we used arrays to evaluate the effect of the number of XML tags in messages (Section 3.3). Also, we measured the performance in a WAN environment (Section 3.4). Finally, we analyzed the overhead of XML-Signature in detail (Section 3.5).

For the evaluation, we used Globus Toolkit 3.2 (GT3.2) on Tomcat 4.1.30 [1]. We chose Tomcat as a service container instead of the standalone container included in Globus Toolkit because it is stated that the standalone container is only for testing purposes and also because our preevaluation shows that it is not as stable as Tomcat. The options for delegation and authorization are set to ”none” throughout the evaluation. The version of XSUL we used is 1.1.5. Other relevant software consists of Sun J2SE 1.4.2_04, and Linux 2.4.21. Both the clients and the server are on the same LAN, connected in 1000 Mbps Full Duplex mode. All the hosts for both the server and clients are of dual Xeon 2.8 GHz with 2048 MB memory.

For the WAN evaluation, we used a host in Matsuoka Laboratory, Tokyo Institute of Technology in Japan as a client. It is of dual Athlon XP 1.2GHz with 896 MB memory. The versions of Globus Toolkit and XSUL are the same. Other relevant software consists of Sun J2SE 1.4.2_04, and Linux 2.4.18. The server is the same as the one for the LAN evaluation. The RTT between the client and the server is 238 msec, and the throughput is approximately 759 Kbps.

3.2. Evaluation with a simple service

This evaluation focuses on the overhead caused by adding security to a service. We used a simple service, echo service, for the evaluation. A client sends a short string (5 bytes) to a server, and the server simply returns the string

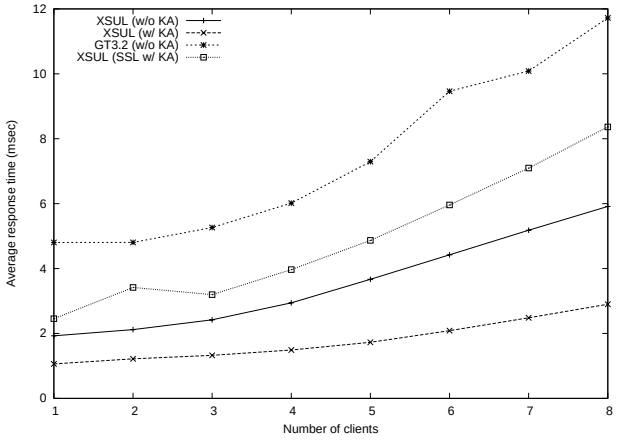

Figure 1. Average response time for echo services without security.

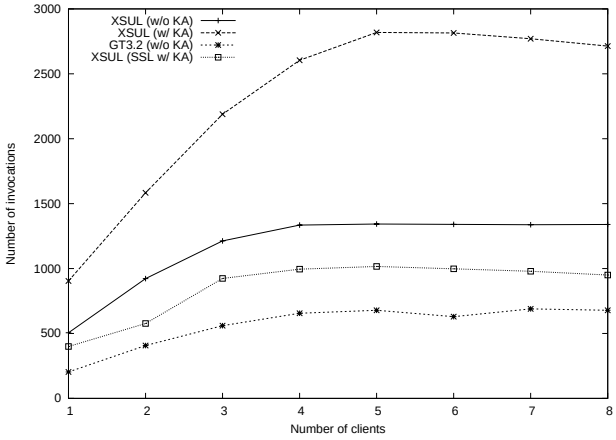

Figure 2. Number of invocations per second for echo services without security.

to the client. During the evaluation, 1 to 8 clients keep invoking the service for 10 minutes for each case. (For initial invocations of WS-SecureConversation, this time was reduced to one minute because of the problem described in Section 3.6.) We measured the average response time of invocations and the number of invocations per second that the server processed. In all of the cases described, the first invocation is not included to omit one time initialization cost, such as class loading, and the performance of the rest of invocations is averaged.

First, for comparison purposes, we studied the performance of the various implementations without security. Figure 1 shows the average response time for echo services. The x-axis is the number of clients invoking the service, and the y-axis is the average response time.

Without the HTTP KeepAlive option, XSUL is approximately twice faster than GT3.2. This is because XSUL manipulates XML with its fast XML Pull Parser [3]. The graph also shows that the HTTP KeepAlive option reduces the response time by half. This graph includes XSUL with SSL, which will be described in greater detail later.

Figure 2 shows the number of invocations that the server processed. The x-axis is the number of clients invoking the service, and the y-axis is the number of invocations per second. This number includes the invocations from all the clients. As the number of clients increases, the number of invocations increases until a certain point. Then, the invocation rate flattens, or sometimes slightly decreases. This number is the maximum number of invocation the server can process.

The graph shows that the server can handle twice as many XSUL invocations as GT3.2 invocations. Also, the HTTP KeepAlive option doubles the number.

Figure 3 shows the average response time for echo services with security. As a whole, transport layer security is faster than message level security. XSUL's SSL is faster than any of the message level security mechanism. We note that this is for one invocation including the cost of the initial handshake. With the HTTP KeepAlive option and multiple invocations, SSL is even faster. It is even comparable with the non-secure versions in Figure 1. GT3.2's transport layer security (httpg) is twice slower than XSUL's. We assume that it is because httpg is no longer supported beyond GT3.2, and hence it was not tuned for performance.

XML-Signature is much slower than transport level security. Compared to the non-secure versions, the difference is almost two orders of magnitude. For XML-Signature, XSUL is slightly slower than GT3.2. Because both GT3.2 and XSUL use the same security library, Apache XML Security [2], to sign and verify XML messages, we were surprised by the difference. We are currently investigating this behavior. For the XSUL implementation of XML-Signature, we also measured the performance with the HTTP KeepAlive option. The result shows that the HTTP KeepAlive option does not improve the performance much. It is because XML-Signature does not involve any handshakes at the beginning. Also, because the overhead cause by XML manipulations is so high that the performance improvement gained by the HTTP KeepAlive option is insignificant.

The graph shows two kinds of numbers for WS-SecureConversation. One is the response time for initial invocations, which includes the time taken for the handshake to establish the connection. The other is for the consecutive invocations after the first invocation, which excludes overhead for the handshake. Compared to GT3.2's implemen-

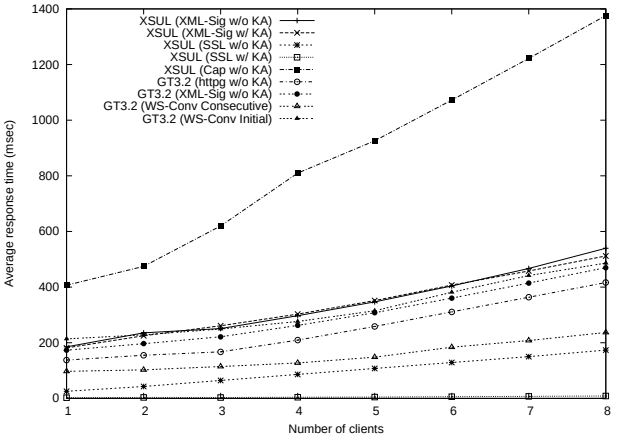

Figure 3. Average response time for echo services with security.

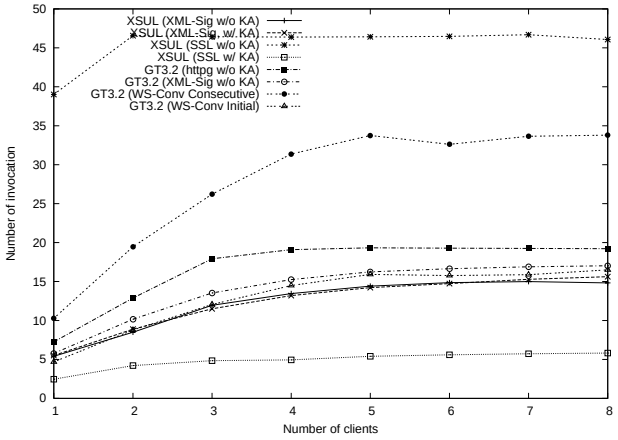

Figure 4. Number of invocations per second for echo services with security.

tation of XML-Signature, the initial invocations are slightly slower. On the other hand, the consecutive invocations are almost half of XML-Signature. This means that if a client invokes the service more than twice, it is worth using WS-SecureConversation instead of XML-Signature. A problem with WS-SecureConversation is that the server needs to keep the context for each client. We will discuss this problem in Section 3.6 in detail.

The capability-based system is more than two times slower than XML-Signature. It is because the capability mechanism needs additional XML manipulations on top of the operations needed for XML-Signature. On the client side, it inserts capability tokens to the SOAP header before it signs a message. On the server side, it verifies the signature for capability tokens on top of the verification of the message itself.

Figure 4 shows the number of invocations per second that the server processed for secure services. Compared to Figure 4, the number of invocations that the server can handle is significantly low. For message level security, it is between 10 to 30. Considering the fact that the server used for the evaluation is relatively powerful, this number is considerably low.

3.3. Evaluation with different array size

This evaluation focuses on the effect of the number of XML tags in messages. We again used echo service, but the input is an array of strings. A client sends an array of strings (each string is 8 bytes long) to a server, and the server simply returns the array of strings to the client. We measured the response time of invocations with different array size. The performance of 20 invocations is averaged. Again, the first invocation is not included to omit one time initialization cost.

Figure 5 shows the average response time for different array size. The x-axis is the array size, and the y-axis is the average response time. As the array size increases, the overhead for transport security becomes relatively smaller. This is because, as the total invocation time increases, the overhead for the initial handshake becomes negligible. Once the secure context is established, the signing using symmetric key does not introduce much performance penalty. For the same reason, the advantage of using the HTTP KeepAlive option is reduced as the array size increases.

In contract to transport security, the overhead caused by message level security grows dramatically as the array size increases. It is interesting to see that the performance is implementation dependent (XSUL is faster than GT3.2 by an order of magnitude), and the difference of the security mechanisms becomes insignificant as the array size grows bigger. This is because most of the time is spent for XML manipulations, especially related to the XML canonicalization [9] before the signing and the verification. The initial handshake for WS-SecureConversation becomes negligible as the array size increases. Performance difference between the symmetric cipher for WS-SecureConversation and the asymmetric cipher for XML-Signature is not apparent because the time spent for calculating the signature is small compared to the one for XML manipulations. The capability-based mechanism is also comparable to XML-Signature. This is because the additional signature verification needed for the capability-based mechanisms is only for capability tokens, not for the entire message.

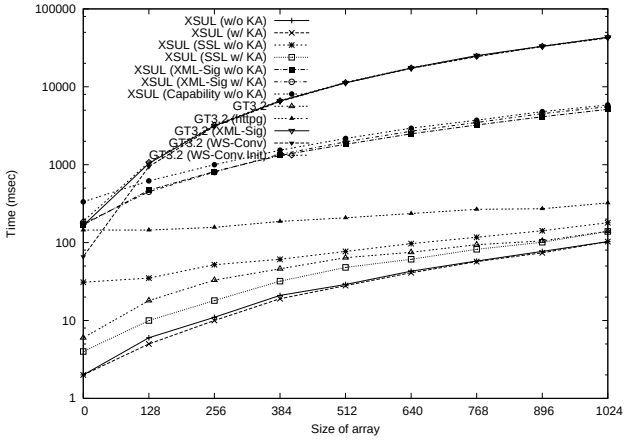

Figure 5. Average response time with different array size

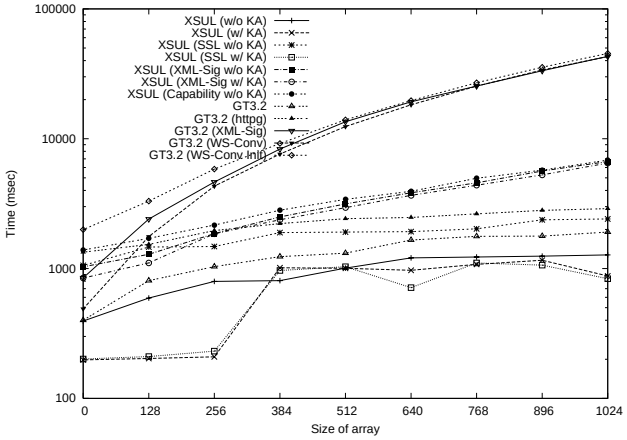

Figure 6. Average response time with different array size (WAN)

3.4. Evaluation in a WAN environment

All of the previous evaluations were performed in the LAN environment. In a WAN environment, some security mechanisms that need initial handshakes, such as SSL and WS-SecureConversation, might become slower because of low network latency. To evaluate the effect of the low latency, we performed the same evaluation as in Section 3.3 in the WAN environment described in Section 3.1.

Figure 6 shows the result of the evaluation. We can observe a couple of distinctive behaviors in the WAN environment, especially with small array size. First, the HTTP KeepAlive option is more effective. It omits the overhead caused by TCP handshake, which is a big overhead in a low-latency network. Especially, in case of SSL, the effect of the HTTP KeepAlive option is significant because it also omits the initial handshake of SSL. Second, the SSL mechanisms (both of XSUL and GT3.2) without the HTTP KeepAlive option are relatively slower in the WAN environment. They are even slower than the XML-Signature mechanisms when the array size is small. The reason is, of course, the big overhead of the initial handshake due to the big RTT. For the same reason, the initial invocations of WS-SecureConversation are slower in the WAN environment.

Note that this evaluation is performed in the environment with extremely low latency. It is across the Pacific Ocean, and the RTT between the client and the server is 238 msec. In a WAN environment with smaller RTT, the result is expected to be closer to the one in the LAN environment.

3.5. XML-Signature break down

For the future improvement of XML-Signature performance, we measured the time taken for each phase to investigate the bottleneck. Since the signing and the verification of a message are the only additional processes added for XML-Signature. We broke down these two processes. For the evaluation, we used the XSUL implementation because XSUL is more tuned for XML handling and has better performance. The service used for the measurement is echo service with array size 1024. The performance of 100 invocations is averaged. Again, the first invocation is not included to omit one time initialization cost.

Table 1 shows the time taken for each phase. The most of the time is spent for the canonicalization of XML (1391.2 msec out of 1542.6 msec for signing, and 1395.5 msec out of 1445.8 msec for verification). To calculate a signature, first, the digest of the message body is calculated using SHA-1 algorithm [5]. This phase takes less than 0.1 msec, and is not included in the table. Then, the digest along with various information related to signing is used to calculate the signature. An asymmetric cipher algorithm based on RSA [7] is used in this phase. However, it only takes 2.2 msec for signing, and 0.4 msec for verification because the size of the data signed is small.

XSUL uses XML Pull Parser for XML parsing and the internal XML representation. To use a third party security library, Apache XML Security, for signing and verification, it needs to convert the internal representation to a DOM tree. This conversion takes 7.7 msec for signing, and 10.2 msec for verification. In case of the signing, after inserting the signature to the SOAP header, it needs to convert the DOM tree back to the XML Pull Parser representation. This takes

Table 1. XML-Signature break down

	(msec)	Phase	(msec)
Signing	1542.6	Conv. to DOM	7.7
		Canonicalization	1391.2
		Signature calculation	2.2
		Conv. from DOM	127.0
		Other	14.5
Verification	1445.8	Conv. to DOM	10.2
		Cert. path validation	22.2
		Canonicalization	1395.5
		Signature verification	0.4
		Other	17.4

more time (124.9 msec) than the conversion from the XML Pull Parser representation to the DOM tree. It is because the canonicalization increases the size of XML documents, making the conversion task more complicated. Note that even with the cost of the conversion of the XML representation, it is obvious that usage of XML Pull Parser has a big advantage (Figure 5).

3.6. A WS-SecureConversation issue

With WS-SecureConversation, a server needs to hold a state for each client. Because it is message level security, even though the transport level connection (TCP connection) is closed, a server still needs to keep the state for feature connections. There is no explicit way to end the message level connection. If a server is accessed by a large number of clients, the amount of states the server needs to hold will be huge.

Figure 7 shows the response time for each invocation. The x-axis is the number of invocations, and the y-axis is the response time for each invocation. One client keeps invoking a service on the server with the WS-SecureConversation mechanism, but each invocation sets up a new connection. As the result, the server has to hold large amount of connection status information. The response time is stable until the 1380th invocation. However, after that the time increases dramatically. The graph stops at the 1680th invocation because the server failed to accept any more connections from the clients by throwing out-of-memory exceptions.

4. Discussion

As a whole, transport level security, SSL, is the fastest. If a client and a server need multiple interactions, it benefits performance most by using it with the HTTP KeepAlive option, but even without the HTTP KeepAlive option, SSL

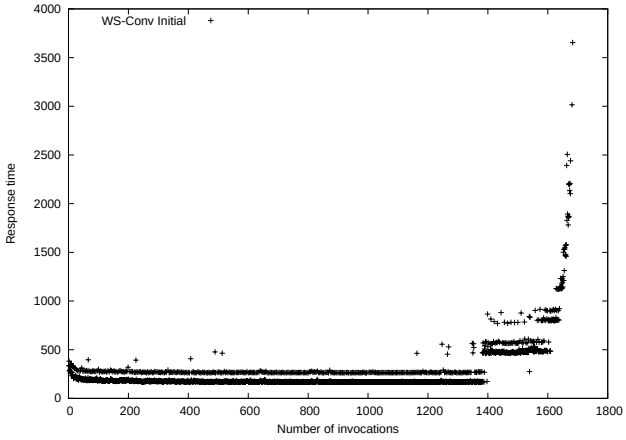

Figure 7. Response time of WS-SecureConversation with consecutive new connections.

is still faster than any other message level security mechanisms. The drawback of SSL is that it is implemented in the transport layer. Hence, it does not work for a connection that includes multiple hops. Also, Unlike the XML-based security mechanisms, SSL does not have the ability to sign only specific portions of a message. Even though usage of this feature is not common currently, it might become important as Grid services mature. For example, a service might need to access some other services and encapsulate the results, which is signed by each service, to the reply message to the client. For those rich features of XML-based security, Globus Toolkit encourages users to use message level security. Transport level security is not supported beyond GT3.2. However, at this moment, most of the services do not support those features. Thus, transport level security can be used in most of the cases. We should have an option to use transport level security if performance is important.

Message level security is generally slower than transport level security. The difference becomes significant as the array size increases. This is because signing of XML messages involves XML manipulations. In particular, XML messages have to be canonicalized before they are signed or verified. The current implementation of XML canonicalizer uses a DOM parser to canonicalize XML documents. Thus, it does not support stream processing, which prevents pipelining each step of the invocation process. This is unfortunate. The XSUL implementation of XML-Signature minimizes the cost of XML manipulations with its fast XML Pull Parser except the XML canonicalization, which XSUL rely on a third party library, Apache XML Security. As the result, the most of the overhead for XML-Signature

is caused by the XML canonicalization. We expect great improvement of the performance by optimizing the XML canonicalization. The possible optimization is to implement a stream XML canonicalizer, or to directly convert the internal XML representation to the canonicalized XML string.

The initial invocations of WS-SecureConversation turned out to be faster than expected. They are only slightly slower than XML-Signature. The consecutive invocations are almost half the cost of XML-Signature for small messages. This means that if a client exchanges messages more than once, WS-SecureConversation should be used instead of XML-Signature. Even if users use WS-SecureConversation instead of XML-Signature for one message exchange, the overhead is still acceptable.

The number of invocations with security that a server can handle is smaller than without security by almost two orders of magnitude. In the case of the server we used for the evaluation, which is a relatively fast machine, it can handle only 30 invocations per second. This indicates that this methods are not suitable for services that a huge number of clients access. In the case of those services, one might want to think of some alternatives, such as to cluster the servers.

5. Conclusion

We introduced several security mechanisms used to add security to Grid services. Our performance evaluation showed that SSL is faster than message level security, and should be used if message level security is not required. If a client invokes a service multiple times, it should also use the HTTP KeepAlive option. Among message level security, WS-SecureConversation should be used. For one-time invocations, XML-Signature is faster than WS-SecureConversation, but even if WS-SecureConversation is used, the performance penalty is negligible. However, WS-SecureConversation has a scalability problem if it is used for a service that is accessed by a huge number of clients.

References

[1] Apache Tomcat. http://jakarta.apache.org/tomcat/.

[2] Apache XML Security. http://xml.apache.org/security/.

[3] Common API for XML Pull Parsing. http://www.xmlpull.org/.

[4] WS/XSUL: Web Services/XML Services Utility Library. http://www.extreme.indiana.edu/xgws/xsul/.

[5] FIPS PUB 180-1. Secure Hash Standard. US. Department of Commerce/National Institute of Standards and Technology, May 1993.

[6] ITU Recommendation X.509 version 3. Information Technology - Open Systems Interconnection - The Directory Authentication Framework, August 1997.

[7] RFC 2437. PKCS #1: RSA Cryptography Specifications Version 2.0, October 1998.

[8] Simple Object Access Protocol (SOAP) 1.1, W3C, May 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[9] Canonical XML Version 1.0, W3C, March 2001. http://www.w3.org/TR/xml-c14n.

[10] XML-Signature Syntax and Processing, W3C, February 2002. http://www.w3.org/TR/xmldsig-core/.

[11] Security Assertion Markup Language (SAML) Version 1.1, OASIS, August 2003.

[12] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[13] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[14] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL Protocol Version 3.0, November 1996.

[15] IBM, Microsoft, RSA Security, and VeriSign. Web Services Secure Conversation Language (WS-SecureConversation) Version 1.0, December 2002.

[16] IBM, Microsoft, and VeriSign. Web Services Security (WS-Security) Version 1.0, April 2002.