WS-I* Compliant Web Service SOAP Message Security Performance

WS-I* compliant web service SOAP message security performance

Gerard McHale and John O'Raw

Department of Computing,
Letterkenny Institute of Technology,
Letterkenny, Ireland
E-mail: gerardmchale1@gmail.com
E-mail: john.oraw@lyit.ie

Kevin Curran*

Faculty of Computing and Engineering,
University of Ulster,
BT48 7JK, Derry, Northern Ireland
E-mail: kj.curran@ulster.ac.uk
*Corresponding author

Abstract: The OASIS web services security (WSS) standard has been developed to provide encryption and digital signing for SOAP messaging to ensure the information in the message is confidential and that the sender and receiver are who they say they are. It has also introduced interoperability and performance problems. Interoperability has been improved with the introduction of the WS-I* Basic and Basic Security Profiles. New web stacks such as Apache CXF have attempted to address performance issues. The purpose of this research is to investigate the performance impacts of securing WS-I* compliant SOAP messages when using the Apache CXF web service framework. We measured the performance impact of WS-Security and WS-SecureConversation under different conditions and using various WS-I* compliant cryptographic algorithms. We found that WS-SecureConversation is the better option when sending a large number of messages but for a small number of large messages WS-Security can sometimes be the better option.

Keywords: web services; WS-I* Basic; security; SOAP; WS-Security; Apache CXF.

Reference to this paper should be made as follows: McHale, G., O'Raw, J. and Curran, K. (2012) 'WS-I* compliant web service SOAP message security performance', *Int. J. Web Science*, Vol. 1, No. 4, pp.291–314.

Biographical notes: Gerard McHale is a graduate in Computer Science of the Letterkenny Institute of Technology. He is currently employed in the Irish IT sector and his research interests include web service security.

John O'Raw is a Senior Lecturer in Computer Science at the Letterkenny Institute of Technology. His research interests are in security and networks.

Kevin Curran is a Reader in Computer Science at the University of Ulster, Northern Ireland. His research interests include indoor location navigation on network security.

1 Introduction

The internet has become an invaluable tool, providing different applications for e-commerce, entertainment, information, communication, etc. Details on the actual number of active domains can vary but the number is somewhere between 124 million and 200 million (Verisign, 2010; Domain Tools, 2010). The popularity of the internet has led to an explosion in the number of malicious attacks against internet and web applications. Statistics show that the number of defacement attacks alone has increased from over 41,000 in April 2008 to over 60,000 in April 2009 up to over 95,000 in April 2010 (Almeida, 2010). Of more concern are malicious attacks on Enterprise IT infrastructures which have become a more serious threat with the growing importance of the internet to enterprise. These attacks can lead to monetary penalties and loss of customer base and goodwill. An important area of web security is to protect the information passed between web applications and web services. These messages are sent across the internet so the information within the messages is accessible to anyone with the knowledge to access it. The WS-Security OASIS standard has been developed to provide encryption and digital signing for SOAP messaging to ensure the information in the message is confidential and that the sender and receiver are who they say they are. The use of web services has become increasing prevalent within enterprise application development with a lot of the larger software companies hosting numerous web services in the cloud. Examples of this are Amazon Web Services and Google Maps API Web Services. Security is especially important when dealing with web services due to the fact that information is being sent across the internet which is inherently unsecure. New methods of securing information within messages have been developed specifically for use with web services. The large number of approaches available for implementing security has also led to interoperability issues between web services.

As yet there is no definitive solution to this problem. During development a balance has to be reached between the level of security necessary to protect the information and the performance requirements. These decisions must be made as part of the requirements gathering phase of any development project. There have been tests and measurements done over the last number of years which have shown the large impact of implementing security in SOAP messaging however there have been advances in web service frameworks and improved Java APIs since these tests. For example the results measured by Liu et al. (2005) were achieved using Java 1.4.2. Apache Axis 1.3 and Java 1.5 were used by Moralis et al. (2007, 2009).

There is a high performance overhead associated with the use of WS-Security in SOAP messages (Liu et al., 2005; Tang et al., 2006; Chen et al., 2007; Van Engelen and Zhang, 2008). The level of overhead depends on the security schemes chosen. Van Engelen and Zhang (2008) determined that using WS Security within SOAP messaging can be a factor of 100 slower than non-secured SOAP/XML messaging. Implementing SOAP security consists of two main tasks which can be described as cryptographic operations and XML processing (Liu et al., 2005). Cryptographic operations can be broken down into signing/verification and encryption/decryption. Signing refers to applying a digital signature to the SOAP message using XML-Signature processing (Bartel et al., 2008) to prove the authenticity of the message. Verification is the process of authenticating the sender of an already signed XML document. Encryption on the other hand, is the process of encrypting the contents of the message so that it cannot be read by anyone other than the intended recipient and is done using

XML-Encryption (Imamura et al., 2002). Decryption is then used to read the contents of the encrypted SOAP message. Previous studies by Liu et al. (2005), Tang et al. (2006) and Chen et al. (2007) have shown that XML signature and XML encryption add a significant overhead to SOAP messages, as much as a 1,500% increase in the response time in some cases. Both the CXF and Metro web service frameworks have been developed since and based on measurements by Sosnoski (2010) should provide improved performance handling. These studies demonstrated that signing was more expensive than encryption, especially when dealing with large numbers of messages, rather than fewer large messages. This is because the size of the message has no impact when signing. Each message must be signed so more signing is done when more messages are sent. They also demonstrated that both signing and encrypting a SOAP message was two to seven times slower than signing alone.

There are significant performance improvements that can be achieved when using the Apache CXF framework compared with the Apache Axis Framework (Sosnoski, 2010). This study considered the affects of using the WS-I* Basic Profile and Basic Security Profile on the performance of WS-Security. The aim in this research was to test the performance impact of WS-Security when applied using the Apache CXF web service framework and Java.

2 Web service security

Transport Layer Security (TLS) and its predecessor Security Socket Layer (SSL) are cryptographic protocols that provide security between web applications across the internet by using HTTP over TLS. TLS provides encryption, authentication and integrity to enable secure point-to-point message exchange between the applications. The goals of the TLS/SSL protocol, as described in RFC5246 are cryptographic security, interoperability, extensibility and relative efficiency. The cryptographic security is provided using a number of different protocols including the TLS Record Protocol, the TLS Handshaking Protocols and encryption algorithms such as RSA and Diffie-Hellman. Interoperability allows independent applications using TLS exchange security parameters without knowledge of the code in other applications. Extensibility allows new encryption methods to be incorporated easily into the framework without major re-work (Dierks and Rescorla, 2008). The final goal of TLS/SSL is to provide relative efficiency. Due to the extra computation associated with cryptography there is a performance cost when using TLS/SSL. Due to this cost the use of TSL/SSL is limited to security critical domains. As of January 2009, the number of TSL/SSL enabled websites was over one million representing only 0.5% of the total domains on the internet today, based on domain counts from Verisign (2010) and Domain Tools (2010). There is a relatively low uptake on the use of TLS/SSL, mainly due to the associated performance overhead. A high end CPU core can handle approximately 1,000 HTTPS transactions per second with 1,024-bit RSA while the same core can handle 10,000 plain HTTP transactions per second (Jang et al., 2010). The most expensive performance cost in TLS/SSL is the RSA computations (Coarfa et al., 2006). Performance improvements can be gained by using Graphics Performance Units (GPUs) to handle the RSA throughput (Jang et al., 2010) but this is still in the initial research stages. The performance overhead associated with TLS/SSL is still far lower than with WS-Security.

A web service is a unit of managed code that can be invoked using HTTP. Web service frameworks have emerged as a means of publishing reusable functionality in which applications are exposed as services or business processes both within a single enterprise and across enterprise boundaries. With this approach applications can be developed by combining the business processes provided by already existing web services. This promotes the reuse of existing functionality and services when creating new services. Applications in a service oriented architecture (SOA) must declare their requirements and capabilities in an agreed, machine-readable format. Web services do this using WSDL. This allows any web service to use the capabilities of any other web service by binding the web services together. SOAP is the communication protocol used for sending messages between web applications. SOAP messages are based on XML. These specifications ensure that the web services architecture allows the integration of distributed and heterogeneous web applications that are independent of programming language, operating system and hardware. This architecture promotes loose coupling between the service provider and the consumer. One of the core functionalities of SOAP is to provide extensibility (Haas, 2003). This extensibility ensures that the SOAP protocol is easily maintainable and can avoid becoming obsolete too quickly. Interoperability is an essential part of enterprise web development. The Web Services Interoperability Organisation (WS-I) was established to promote interoperability. The goal of the WS-I is to establish best practices for web services interoperability, for selected groups of web services standards, across platforms, OSs and programming languages (WS-I, 2010). WS-I has produced a number of profiles to promote interoperability. The profile of interest during this study was the WS-I Basic Security Profile 1.1 (WS-I, 2010) which is specifically related to standardising the WS-Security protocol within SOAP messages to improve interoperability between web services.

TLS and SSL provide very effective point-to-point message security. In the case of web service communication where only one hop is required TLS/SSL can be used effectively to provide confidentiality, authentication and integrity. Applying TLS/SSL as opposed to WS-Security can also significantly reduce the complexity and the performance overhead (Shirasuna et al., 2004). Web service messaging has the capability to traverse multiple applications and intermediary nodes before reaching an end point. TLS/SSL prevents message re-routing because the addressing information cannot be read within the message header at the intermediary nodes (Moralis et al., 2009). WS-Security on the other hand can provide end-to-end message level security (Web Services Security, 2006) and also allows cryptography to be applied to separate parts of the SOAP message. Therefore different combinations of body blocks, header blocks, etc. can be encrypted which allows intermediary nodes access certain parts of the message only (WSS, 2006). The full message can then be decrypted at the final destination. This provides end-to-end security which is not available when using TLS/SSL. SOAP was developed to exchange messages over a variety of underlying protocols. TLS/SSL is based specifically on the HTTP transport protocol. The capacity and application of web services would be limited if its security relied solely on this transport dependant technology. The mechanisms provided by WS-Security are applied directly to the SOAP message so the underlying protocol used has no impact on the security of the SOAP message (WSS, 2006). The release of SOAP version 1.2 further improved on the SOAP protocol. It was cleaner, faster, and more versatile and provided better web integration than its predecessor (Haas, 2003).

There are 16 different algorithm suites supported by the WS-SecurityPolicy, all of which can be used for the encryption and signing of SOAP messages. The use of these is determined by the selection of a particular algorithm suite within the WSDL. However, of these 16 possible suites only four are recommended by the WS-I* Basic Security Profile (WS-I, 2010). Although the algorithm suites are not specifically mentioned in the profile the recommended algorithms and message digests are included. This information was used to determine the suites supported by WS-I*. The algorithm suites supported are Basic256, Basic128, TripleDes and TripleDesRsa15. Ichikawa et al. (2000) demonstrated that the RSA AES algorithm was faster than the DES algorithm and by extension also faster than the TRIPLEDES algorithm. It also provided enhanced security over the TRIPLEDES algorithm. Srirama et al. (2007) showed that there was only a slight difference in the transmission latency when using the encryption algorithms TRIPLEDES, AES128 and AES256, with TRIPLEDES providing the worst performance in most cases. These measurements used the asymmetric RSA 1024 algorithm to exchange the symmetric keys. These tests were done for mobile web service so this could possible explain the different findings between the two studies.

2.1 Transport layer security and secure socket layer security

The WSS (2006) provides three main mechanisms for providing security within SOAP messages. These mechanisms are Security Tokens, XML Signature (Bartel et al., 2008) and XML Encryption (Imamura et al., 2002) which provide authentication, integrity, non-repudiation and encryption. These mechanisms are the building blocks used to create secure technologies. However, these mechanisms alone cannot guarantee security. The intent of the Basic Security Profile is to promote interoperability. While it does not define best security practices or guarantee security, it does use known security weaknesses to reduce choice and thus improve interoperability and security. This has the effect of improving message security if the standard is followed. In the end however, full security can only be guaranteed by implementing the required security mechanisms in the correct way. This handling is left to each particular application implementing web service security.

SOAP messages can be signed and encrypted by multiple entities signing and encrypting overlapping elements. A different processing order can give very different results. For example, with signature before encryption the signer is known to have created or vouched for the plain text but there is no way to know whether the signer performed the encryption. Alternatively with encryption before signing, the signer is known to have vouched for the cipher text but it is not known by the receiver if the signer was aware of the plaintext. The Basic Security Profile defines the order for processing elements within the security headers. This ensures the receiver of the message will achieve the correct result during deciphering of the message if the order is followed. Davis (2001) described security holes and potential solutions with both of these approaches. In a production environment the recommendations by Davis should be followed. However, it was decided that this was not necessary in this case. For a single message that will be sent between a web service and a client, via a number of routers, WS-Security is the best approach for implementing security. This allows the use of numerous different security tokens including username, X.509 and Kerberos tokens for the encryption and signing of the XML SOAP messages. The OASIS standard Web

Services Security: SOAP Message Security 1.1 (WSS, 2006) describes the format of SOAP messages when using WS-Security. Although possible to handle the encryption using synchronous keys it is recommended that asynchronous tokens such as X.509 or Kerberos are used. For cases where multiple messages are expected between the client and the web service, WS-SecureConversation 1.4 (2009) is the preferred approach. This is because the asynchronous approach is only used to share a synchronous key between the client and the server. After that the synchronous key is used for encryption of the message. This produces less overhead than using asynchronous keys for all the messages transmitted.

2.2 WS security and SOAP messaging performance

Real world applications need to use both encryption and signing. However, there is a performance cost associated with this as shown by Liu et al. (2005) and Tang et al. (2006). If performance is an issue then possible options may be to keep the messages as small as possible or only sign and encrypt the sensitive parts of the message. XML processing is the second area that has a large impact on the performance of WS-Security. This includes parsing, validation, transformation and document tree traversal of the XML SOAP messages. Exclusive XML canonicalisation (Boyer et al., 2002) is the process of converting an XML document into a standard form. This is particularly important for XML-Signature because it removes issues caused by the flexibility of XML, which allows a document to be changed in certain ways but still be considered the same. For example, an XML document is still considered syntactically the same if additional white spaces are added. However, this would change the signature of the document so would affect the signing process in WS-Security. Van Engelen and Zhang (2008) show the performance impact associated with XML canonicalisation in some detail. The impact of XML processing and canonicalisation has not been considered for this research.

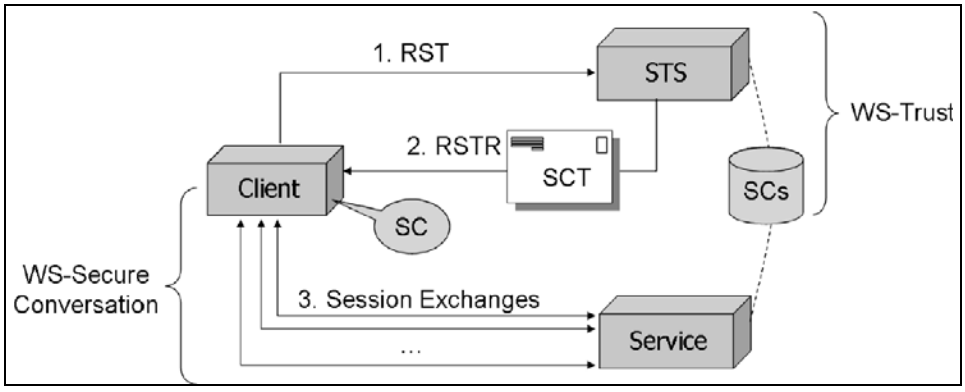
The use of different security tokens can have an impact on the performance overhead of using WS-Security. Most of the studies done on this topic have based the measurements around the username and X.509 tokens. The username token is a symmetric key which provides a lower level of security than asymmetric keys such as X.509. For that reason it was not considered during this research. Tests completed by Moralis et al. (2007) compared the X.509 and Kerberos security tokens. The results showed that Kerberos can handle approximately 15–20% more messages per second than the X.509 certificate over a range of message sizes from 60 to 800 bytes. These tests also showed that Kerberos displayed up to a 28% packet throughput improvement over X.509 under full load conditions on the server. Further studies by Moralis et al. (2009) demonstrated that Kerberos consistently exhibited up to 50% message throughput improvement over X.509. However, it was shown by Van Engelen and Zhang (2008) that WS-Security message encryption and signing using HMAC symmetric keys can be an order of magnitude faster than using Kerberos, assuming the X.509 token profile is used. The disadvantage of using HMAC is that symmetric keys require a shared secret. This is not simple with SOAP messaging because unlike TLS, there is no handshake protocol in WS-Security. Van Engelen and Zhang (2008) showed how WS-SecureConversation could be used to overcome this problem.

2.3 WS secure conversation and SOAP messaging performance

WS-SecureConversation is an OASIS specification which extends WS-Security (WS-SecureConversation 1.4, 2009). It enables the exchange of a symmetric key between a SOAP client and web service in an initial handshake message. This symmetric key can then be used to secure ongoing web service message exchanges with less processing overhead than WS-Security. It is a similar pattern to that used during a TLS handshake. The disadvantage of using WS-SecureConversation is the extra complexity and dependencies introduced into the system. Additional WS-Trust and WS-SecureConversation elements must be included in the WSDL and the SOAP messages to support the secure conversation. A decision on whether or not to use WS-SecureConversation would have to weigh the performance benefits against this additional complexity.

The initial handshake between the client and the web service is handled by WS-Trust which also builds on top of WS-Security. It relies on a security token service (STS) to evaluate requests and issue a security context token (SCT). The issued SCT is then used to secure the messages during WS-SecureConversation. The most common usage of WS-SecureConversation is for the client to communicate with a single server. In this situation the STS and the web service can be co-located on the same server and therefore both have access to the one security context (SC). This was the approach taken during this research. The steps necessary for using WS-SecureConversation can be described with the aid of the diagram shown in Figure 1 (Bhargavan et al., 2007). At step 1 the client contacts the STS with a request security token (RST) message. At step 2 the STS generates a new SC, caches it, and replies to the client with a new request security token response (RSTR) message which includes an SCT. The SCT contains enough information for the client to calculate the same SC. This allows the client and server to exchange messages which are protected using the keys derived from the shared secret of the SC (step 3). Although it is possible to use the SCT for signing and encryption, it is recommended by the WS-SecureConversation 1.4 (2009) standard that "derived keys be used for signing and encrypting messages associated only with the security context".

Figure 1 Protocol using WS-Trust and WS-SecureConversation



Experiments by Liu et al. (2005) have determined that the use of WS-SecureConversation is noticeably faster than using normal WS-Security. Further research by Van Engelen and Zhang (2008) on the performance impact of using WS-Security to sign and verify SOAP messages determined that the use of the symmetric HMAC algorithm was approximately seven times faster than DSA and eight times faster than RSA which are both asymmetric algorithms. The symmetric HMAC algorithm can only be used in conjunction with WS-SecureConversation because the symmetric algorithm must be agreed between the client and web service before use. This is done during the initial handshake. However, once the handshake has been completed the results show an order of magnitude performance improvement for all subsequent messages when using HMAC as opposed to the public key algorithms DSA/RSA, which must be used when only using WS-Security.

3 Performance framework

Based on similar experiments in the past (Moralis et al., 2009; Tang et al., 2006; Liu et al., 2005) it was decided that the use of a simple web service and client was the best option for this research. This approach meant that other processing overheads that might distort the performance measurements could be ignored. All the testing was completed on a Pentium Dual Core 2.3GHz CPU with 4GB of RAM running a 32-bit Windows 7 Operating System. The Java version used was JDK 1.6.0_24. The Apache CXF web service framework using JAX-WS was used to handle the communication between the web service and the client. The BouncyCastle (2011) JCE provider was used to handle all the cryptographic functions. The jar file needed for this (i.e., *bcprov... .jar*) was included as part of the Apache CXF 2.3.3 installation. To remove the possible impact of results being affected by network latency all the tests were run with both the client and the web service hosted on the same machine. Both the web service and the client were kept small to reduce the possible impact of reduced CPU processing available for XML encryption and signing caused by the garbage collection of both the client and web service running side-by-side. To add some complexity to the web service a database was initially included as part of the web service. The MySQL database (version 5.5.8) was selected because it is a very popular and well documented open source database. The database was installed using the 32-bit .msi file. For initial development the standard configuration was selected during the installation process. The database was divided into the three tables. In each table the ID column was set as the primary key. The BOOKS table contained a list of all the books available to the application. The AUTHORS table contained all the authors used in the application and the PUBLISHERS table contained a list of all the publishers.

A persistence layer was used to de-couple the persistent database of an application from the business logic. The use of a good persistence layer makes an application more maintainable. For example the database used in an application could be updated or completely changed without an impact to the business logic of the application. Hibernate is a popular and well documented open source persistence framework used primarily with Java applications. It was chosen to handle the persistence layer and the object relational mapping between the Java business logic and the relational database. For a description of installing, configuring and using Hibernate see King et al. (2011). This allows Java classes to be mapped to the different tables in a relational database. For this research version 3.3.2 of Hibernate was used. Version 3.4.0 of hibernate annotations was also

used. A separate class was created to represent each of the tables within the database. These classes were *Book.java*, *Author.java* and *Publisher.java*. Annotations were used within each class to map the java classes to the corresponding database tables. To promote loose coupling in the application the javax.persistence annotations were used, rather than the Hibernate specific annotations. The benefit of this was that a different persistence framework could be incorporated into the application without having to change the java code. The @Entity annotation was used to signify each class as an entity that would be mapped to the database and the @Table annotation was used to determine the specific table in the database. The @Column annotation was used to provide the name and other details of each column in the table. Note that in most cases each private field in the Java classes corresponded to a column in the corresponding database. The @Id annotation was used to determine the primary key for each table and the @Generated annotation ensured that the primary key was incremented sequentially each time a new record was added to the table. Finally, the @ManyToOne annotation was used to create links between the different tables. For example, this annotation was used to associate multiple books with a single author. It was also necessary to create a hibernate.cfg configuration file which was used to configure hibernate. The database connection settings such as the location of the database and the username password needed to access the database were set here. All the relevant classes corresponding to the database tables were mapped. The framework was also told which SQL language it should be using to access the database.

3.1 Web service frameworks

Measuring the performance impact of using WS-Security and WS-SecureConversation was the main objective of this research therefore a number of test scenarios were developed to measure the performance impact. When developing the test scenarios a number of issues were considered. The first question was whether the client and the web service should both be run on the same machine. Liu et al. (2005) and Van Engelen and Zhang (2008) performed all the tests with both the client and the web service running on a single machine. Tang et al. (2006) and Chen et al. (2007) chose to host the client and the web service on separate machines while Moralis et al. (2007) and Shirasuna et al. (2004) had multiple clients, each located on a separate machine. These approaches provided a more realistic real world scenario and affected the measurements due to network latency. This approach also removed the potential impact of results being affected due to increased garbage collection of both applications running simultaneously on the one machine and allowed the client and server processing overhead to be evaluated separately. However, the benefit of using one machine to host the client and the web service was that it removed the issue of the performance results being affected by network latency, thus generating more consistent results. Juric et al. (2006) showed that performing the same WS-Security tests on a single machine or multiple machines can have a large impact on the results. Therefore it was decided to run the tests with both the client and the service on a single machine so that any measured differences could be related directly to the CPU processing in the client and the web service due to the selection of different web service security options.

Another question was the method that should be used while running the tests. One approach by Shirasuna et al. (2004) was to send the message repeatedly for ten minutes.

The average response time of the invocations and the number of invocations per second that the server processed were both measured. Liu et al. (2005), Juric et al. (2006) and Van Engelen and Zhang (2008) chose to resend each message between 100 and 100,000 times and measure the average CPU time. In all cases the first invocation was ignored from the measurements to omit one time initialisation costs. (Juric et al., 2006) also included the additional step of repeating each test 12 times, discarding the maximal and minimal times and taking the average of the remaining ten messages. It was decided that the best approach was to repeat the same message a set number of times. The response time for the first message was ignored so that the results would not be impacted by initialisation costs. The best performance was taken as the final result. The type of messages sent between the SOAP client and the web service provider was also an important consideration for this research. Different approaches to this question were used in previous performance evaluations of WS security. Van Engelen and Zhang (2008) used EchoString and EchoStringArray SOAP RPC-encoded arrays, which meant that a string or an array of strings was sent to the web service and simply echoed back to the client. Moralis et al. (2007) and Shirasuna et al. (2004) employed a similar approach by developing a web service that accepted a string and echoed the same string back to the sending client. Tang et al. (2006) used an alternative approach where the client sent a request and the web service responded by returning a specific amount of customer records. The approach taken by Tang et al. (2006) was designed to mimic more real world scenarios than that other approach which just echoed the received string back to the client. A similar approach was used in this research where an array of book objects were returned to the client based on the information contained in the request message.

The size and frequency of the response message will also have an impact on the performance due to WS-Security (Sosnoski, 2010). When combining signing and encryption in the SOAP messages Sosnoski showed that the performance impact by WS-Security on a large number of small response messages was ~22 times larger than a plain message. However, the performance impact when using a smaller number of large messages was only ~13 times larger than the plain message. This was due to the overhead included in the header, which was the same regardless of the size of the message body. Due to these findings it was determined that the tests during this research would be repeated for a large number of small messages and a smaller number of large messages. The Java Cryptography Extension (JCE) (2002) provides a framework and implementations for encryption, key generation and key agreement and is necessary to provide security for SOAP messages. In a comparison done by Juric et al. (2006) between three different JCE providers it was discovered that there was very little difference between the performances of the three. The three JCE providers in question were BouncyCastle, Wedgetail JCSI and OpenSource HBCI Toolkit. As Juric et al. (2006), Liu et al. (2005), Moralis et al. (2009) all used the BouncyCastle JCE provider (BouncyCastle, 2011), it was decided that this JCE would be used for this research as well.

Three of the main open source Java web service frameworks available today are Apache Axis2, Sun/Oracle Metro and Apache CXF. All of the web service frameworks support WS-Security and WS-SecureConversation as required here. However, performance tests have shown Apache CXF and Metro to be more than twice as fast as Apache Axis2 when WS-Security is used with XML-signature and XML-encryption (Sosnoski, 2010). These tests also demonstrated that Metro was slightly faster than Apache CXF. Results of similar tests using WS-SecureConversation showed that Metro

provided the best performance (Sosnoski, 2010). Both Metro and Apache CXF were over 40% faster than Apache Axis2 across all the different configurations tested. Metro was approximately 12–15% faster than Apache CXF. It should be noted that these tests were carried out using Metro 2.0 and Apache CXF 2.1.7/2.1.8. The latest versions of Metro and Apache CXF are 2.0.1 and 2.3.2 respectively so these results may not be completely accurate any longer. Sosnoski determined that Apache CXF was the preferred framework for use with WS-Security (2010). This decision was based on a number of experiments where he compared the performance of the three main Java web service frameworks mentioned above. He also considered other issues such as simplicity of configuration. However, this decision was partially based on personal opinion and also considered issues such as simplicity of configuration. In the end there was very little to choose between Apache CXF and Metro. Due to the fact that the performance differences between and Apache CXF were relatively small it was decided that a decision on which framework to use would be made during implementation. During implementation it was discovered that Apache CXF was easier to configure, especially considering that Apache Tomcat was the application server that was being used. Due to this, CXF was chosen as the web service framework for the research.

Version 2.0 of the WSDL standard was released in 2007. Despite the improvements that it introduced over the WSDL 1.1 standard (Christensen et al., 2001) it is still not widely used in the web services industry. As such there are no test tools available yet from the WS-I* group for validating that a WSDL 2.0 conforms to the WS-I* Basic Profile 2.0. A requirement of this research was that the WSDL and SOAP messages used would be WS-I* compliant. While it would be possible to conform to the standards by manually checking each WSDL and SOAP message it was not considered a viable approach. Therefore it was decided that WSDL 1.1 would be used through this research. It is possible to validate WSDL 1.1 using the WS-I* Basic Profile 1.1 profile tool which is bundled with the soapUI test tool. The other issue found during the implementation phase was that the existing test tools for validating WSDL 1.1 did not support the use of SOAP 1.2. This was determined through testing despite the fact that the soapUI test tool claims to support validation of SOAP 1.2. Therefore, for the validation the tool to be of any use it was necessary to use SOAP 1.1 throughout this research. This was not an ideal solution because of the advantages associated with SOAP 1.2 (Haas, 2003).

3.2 Design approach for the WSDL

Java API for XML Web Services (JAX-WS) is a fundamental technology for developing SOAP-based web services (Metro Web Services, 2010). The JAX-WS API was designed to replace the JAX-RPC API and was introduced in JSR 224 (2007) and reflected the move in the industry away from the RPC-style towards document style web services. Therefore JAX-WS has been chosen for this research. JAX-WS uses Java annotations to provide a detailed mapping from a service defined using a WSDL to the Java classes that will implement that service. It allows for complex types defined in the WSDL to be mapped to Java classes using the Java Architecture for XML Binding (JAXB). The Service Endpoint Interface (SEI) is the java code that is exposed to the consumers of the web service. There are a number of different WSDL styles that can be used with JAX-WS web services. Butek (2005) provided a description of each of these styles and when each style might be used. Due to the requirement to be WS-I* compliant the

document/literal wrapped WSDL style was chosen for this research. The recommended approach for developing a new web service is referred to as the 'top down' approach. This involves designing the services using WSDL first and then generating the code to implement the designed services. The benefit of this approach is that the interface is well defined before any development work begins. It also enforces the concept that the web service is implementation neutral. The top down approach was used for developing the web service and client during this research.

3.3 Implementing WS-security using Apache CXF

CXF implements WS-Security by integrating Apache WSS4J. Apache WSS4J is an implementation of the OASIS WS-Security. Within CXF, WSS4J can be configured using either interceptors or WS-SecurityPolicy. However, the use of the WS-SecurityPolicy standard provided a method for configuring and controlling the security requirements of the application using industry agreed standards. Due to the requirement to follow the WS-I* standards it was decided that WS-SecurityPolicy should be used. When implementing WS-Security there are multiple approaches which can be used for authentication. Some of the approaches supported by Apache CXF are username tokens, Kerberos and X.509 certificates. The security provided by username tokens is limited so this authentication protocol was not used during the research. The token analysed during this research was X.509.

The algorithm suite used by the WS-SecurityPolicy standard specified a number of different algorithm combinations that could be used in conjunction with the WSDL. The standard defines 16 supported suites, each offering different digest, encryption, key-wrap and key derivation algorithms, which are described in the WS-SecurityPolicy standard (WS-SecurityPolicy 1.3, 2009). Only some of the suites defined were valid for this research because only certain algorithms are recommended by the WS-I* Basic Security Profile. These were Basic256, Basic128 and TripleDes. Although the WS-I* Basic Security Profile does support SHA256, it recommends the use of SHA1 unless it is not suitable for some reason. Therefore it was decided not to use SHA256 in this research. The WS-I* Basic Security Profile also states that RSA-OAEP must be used for the transport of 128 and 256 bit keys. These recommendations meant that only Basic256, Basic128, TripleDes and TripleDesRsa15were valid for this research. The X.509 standard uses a private key stored on the machine sending the message and a certificate containing a public key which is stored on the machine receiving the message. The public key is then used to decrypt and authenticate the received message. In a production application this certificate would be provided by a certification authority. However, in this research the private key and the certificate containing the public key were generated using Java keytool which manages a keystore of cryptographic keys and trusted certificates.

The first step was to create the private keys for the web service and for the client using the commands shown in Figure 2. The command for the web service generated a key pair and wrapped the public key in an X.509 version 3 self-signed certificate. The certificate was stored in a single-element certificate chain. This certificate chain and the private key were then stored in the new keystore servicekeystore.jks identified by the alias myservicekey. The client's private key and certificate chain containing the public key certificate were stored in the keystore clientkeystore.jks which was identified by the alias myclientkey.

Figure 2 Keytool command to create private key store in web service and client

```
keytool -genkey -alias myservicekey -keyalg RSA -sigalg SHA1withRSA -keypass skpass -storepass sspass -keystore serviceKeystore.jks
-dname "cn=localhost"

keytool -genkey -alias myclientkey -keyalg RSA -sigalg SHA1withRSA -keypass ckpass -storepass cspass -keystore clientKeystore.jks
-dname "cn=localhost"
```

- *alias* identified the created keystore.
- *keyalg* specified the algorithm used when generating the key pair. In this case the RSA algorithm was used.
- *sigalg* specified the algorithm that was used to sign the self-signed certificate. The algorithm used had to be compatible with *keyalg* so SHA1withRSA was selected.
- *keypass* denoted the password used to protect the private key. During this research a simple password was selected. In production a more secure password should be selected.
- *storepass* denoted the password used to protect the keystore. Again, a more secure password should be selected for a production environment and it should be different to the keypass value.
- *keystore* provided the location of the new keystore file that was created.
- *dname*s pecified the X.500 distinguished name that was associated with the alias. This value was then used as the issuer and subject fields in the self-signed certificate.

To set up two way trust between the SOAP client and the web service, the public key certificate of each was added to the keystore of the other. This was done using the commands listed in Figures 3 and 4.

Figure 3 Add trusted certificate from client to service keystore

```
keytool -exportcert -rfc -keystore clientKeystore.jks -storepass cspass -alias myclientkey -file MyClient.cer
keytool -importcert -trustcacerts -keystore serviceKeystore.jks -storepass sspass -alias myclientkey -file MyClient.cer -noprompt
```

Figure 4 Add trusted certificate from service to client keystore

```
keytool -exportcert -rfc -keystore serviceKeystore.jks -storepass sspass -alias myservicekey -file MyService.cer
keytool -importcert -trustcacerts -keystore clientKeystore.jks -storepass cspass -alias myservicekey -file MyService.cer -noprompt
```

The attributes used for both the SOAP client and the service were *rfc* which ensured that the certificate was output in the printable encoding format defined by Internet RFC 1421 and *file* which was used to specify the name of the file where the exported certificate was stored temporarily. The keystore files containing the private key and the public certificate were then added to the src/main/resources folder of each project. Adding the public key certificate of both the SOAP client and the web service provider to the keystore of the other meant that messages encrypted and signed using the private key could be decrypted and authenticated in the receiving node using the appropriate public key. These certificates were self-signed, which was sufficient for this local environment. However, in a production environment it would be necessary to get the exported certificate signed by a valid Certification Authority (CA). The signed certificate would then be added to the keystore of the receiving node. This would ensure that the receiver could trust that the

public key was valid and authentic. When sending or receiving a SOAP message it was necessary to access the private key contained in the keystore for decryption and signing. It was not possible to access the private key from the keystore without providing a password. The same rules apply for both the web service and the SOAP client. This was done by creating a class in both the SOAP client and the web service provider which implemented the CallBackHandler interface. The handle method was called when a request was made to retrieve the password for the private key stored in the keystore (i.e., when signing or decrypting a message). A check was made in the WebServiceCallBackHandler object to determine whether the request was for signing or decryption and if so the keystore password was then stored in the object so that it could be used to access the private key. A similar class was created in the SOAP client. Note that in this class the username and password for the private key were stored in a HashMap object that was created when the class was initially instantiated. In a production application this approach should not be used but it was considered sufficient for this research where it was the performance impact that was important. A number of other details such as correct error handling were also omitted for simplicity.

4 Performance results

We present here the results of the WS-Security and WS-SecureConversation performance evaluation. All the testing was completed on a Pentium Dual Core 2.3GHz CPU with 4GB of RAM running a 32-bit Windows 7 Operating System. The Java version used was JDK 1.6.0_24. The Apache CXF web service framework using JAX-WS was used to handle the communication. The BouncyCastle (2011) JCE provider was used to handle all the cryptographic functions. The jar file needed for this (i.e., *bcprov... .jar*) was included as part of the Apache CXF 2.3.3 installation. To remove the possible impact of results being affected by network latency all the tests were run with both the client and the web service hosted on the same machine. Both the web service and the client were kept small to reduce the possible impact of reduced CPU processing available for XML encryption and signing caused by the garbage collection of both the client and web service running side-by-side. All tests were run using signing and encryption because in a real world scenario it is unlikely that one would be used without the other.

The different technologies actually used in this research were Apache Ant version 1.8.2, Apache CXF version 2.3.3, Apache Tomcat version 6.0.29, BouncyCastle JCE provider version 1.45, Eclipse Helios Java EE version 1.3.1, Eclipse WTP plugin version 3.3.2, Hibernate version 3.3, Hibernate Annotations version 3.4, Java JDK version 1.6.0_24, MySQL version 5.1.15, Java Architecture for XML Binding (JAXB) version 2.2.1, Java API for XML Web Services (JAX-WS) version 2.2, SOAP, Spring version 3.0.5, WS-SecureConversation, WS-Security, WS-SecurityPolicy and WS-Trust.

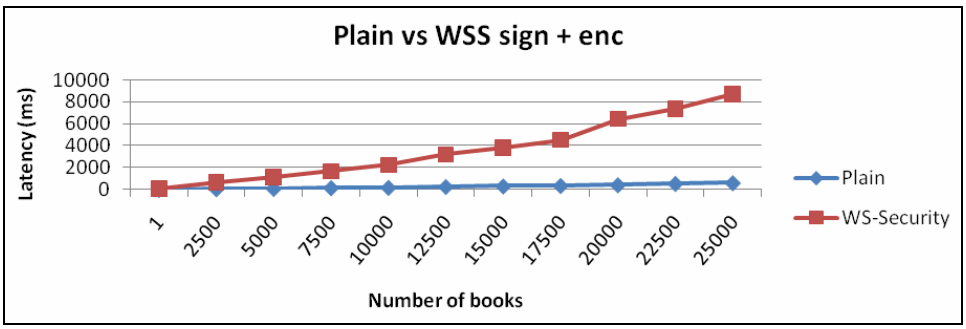
4.1 Analysis of WS-Security for different message sizes

The first test was designed to determine the performance impact of using WS-Security with different message sizes. WSS Encryption and WSS Signature were both included in the SOAP messages because of the necessity of using both in real world applications. All the WSS messages were encrypted and signed using the Basic256 algorithm suite. The

different message sizes were achieved by returning a different number of books in the response message (i.e., number of books = 1, 2,500, 5,000, 7,500, 10,000, 12,500, 15,000, 17,500, 20,000, 22,500, 25,000). Each message size sequence was run multiple times with only the best time for each sequence kept. For each message sequence the response time for the plain SOAP message and the WSS SOAP message were recorded. The results can be seen in Figure 5. The performance impact of using WSS was calculated using the formula:

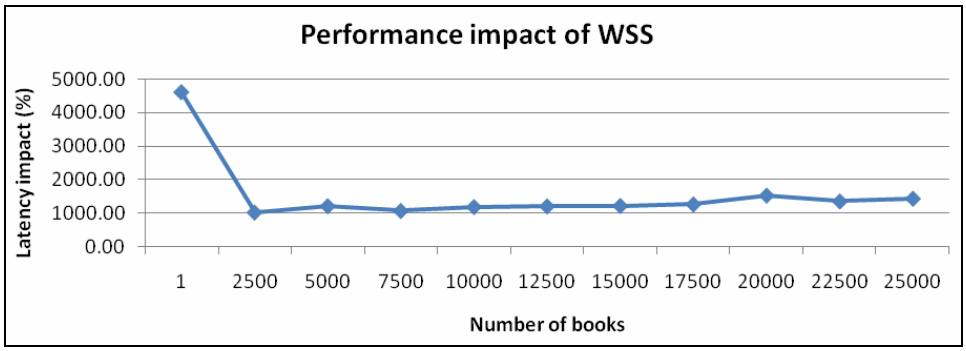
$$Performance\ impact = \frac{SOAP_{wss}}{SOAP_{plain}} * 100$$

Figure 5 WSS SOAP messages compared with plain SOAP messages



The performance impact results based on this calculation are displayed in Figure 6. It was expected that the latency for the SOAP messages would be directly related to the message size and that the latency would increase in a linear fashion as the message size increased. The results from this test agreed with the expected results in almost all cases. The percentage performance impact associated with signing and encrypting a SOAP message containing only one book however was very large compared with the other results. This anomaly can be accounted for by the latency of a single plain SOAP message which came out at approximately 1ms. This low number was understandable considering the small amount of data sent between the client and web service. However, it resulted in a very low divisor in the formula which affected the result.

Figure 6 Percentage increase in latency when using WS-Security



As expected there was a large performance drop when using WS-Security. Ignoring the anomaly with the measurement for one message the performance drop increased slowly as the size of the message increased. The performance impact ranged from a factor of 10 (i.e., 1,000%) to a factor of 15 (i.e., 1,500%) as the message size increased. This can be explained by the additional XML processing, XML canonicalisation and the increased computationally intensive work of generating digests and encrypting data that must be handled as the message size increases.

4.2 Analysis of algorithms supported by WS-I Basic Security Profile*

There are 16 algorithm suites supported by the WS-SecurityPolicy standard (WS-I, 2010). To encourage interoperability the WS-I* Basic Security Profile recommends the use of only four of these suites. The purpose of these test cases was to measure the performance impact of these algorithm suites on WS-Security. The recommended algorithm suites are Basic256, Basic128, TripleDes and TripleDesRsa15.

Figure 7 Comparison of different algorithms when using WS-Security

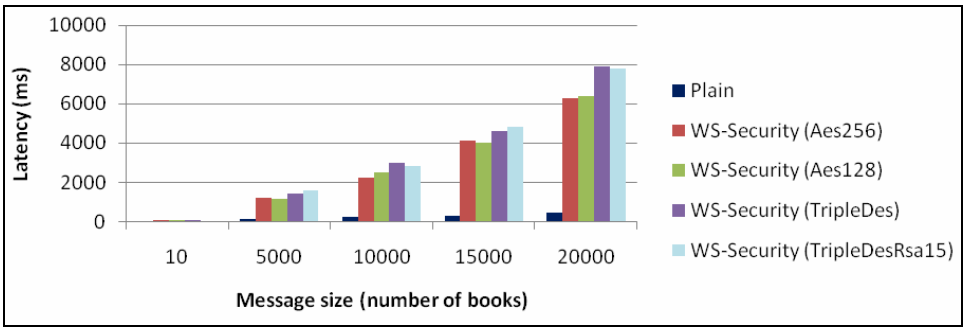
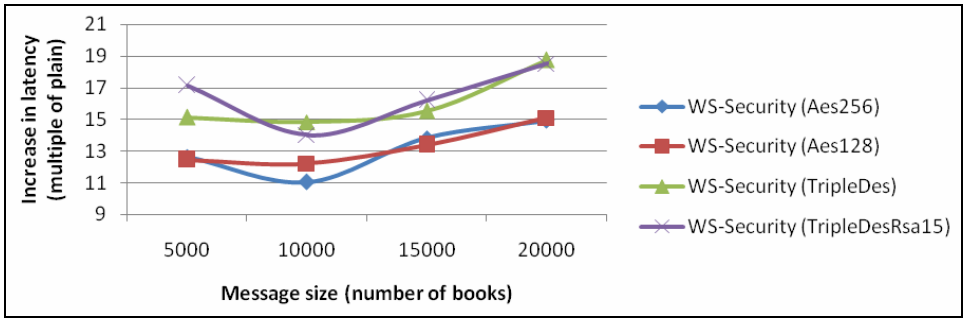


Figure 8 Factor increase in latency compared with a non-secured SOAP message

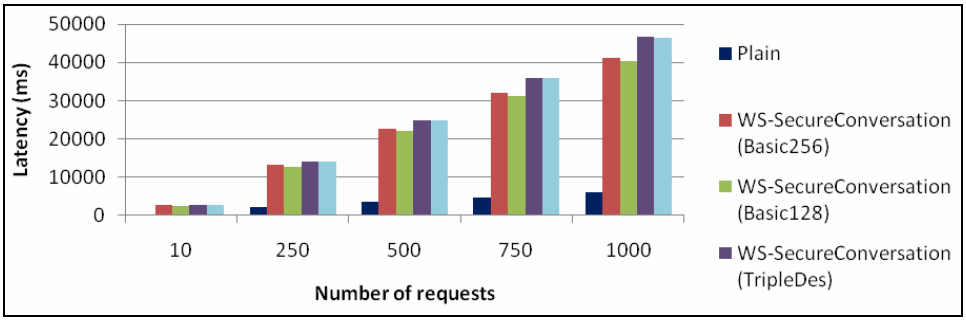


The first test was designed to measure the impact of the different algorithm suites on the performance impact caused by WS-Security. In this test each algorithm suite was used to sign and encrypt a range of different sized messages. The size of each message was determined by the number of books included in the response from the web service (i.e., number of books = 10, 5,000, 10,000, 15,000, 20,000). WS-Security was used to secure the SOAP messages. For each combination of message size and suite a number of

requests were sent. Only the best result was recorded in each. The results of these measurements can be seen in Figures 7 and 8.

These results show that there is a difference when using the different algorithm suites. As expected, based on the study by Ichikawa et al. (2000) the Basic128 and Basic256 suites provided better performance than the TripleDes and TripleDesRsa15 suites. The basic suites had a performance cost of that was a factor between 11 and 15 of a plain SOAP message. The TripleDes algorithms on the other hand ranged between 14 and 19. This is a major difference, especially considering the improved security provided by AES (used in the Basic suites) over TRIPLEDES. The results of the comparison between the Basic256 and Basic128 however were unexpected. The initial expectation was that Basic128 would have less impact on performance than Basic256. The results show that this is not the case with the impact of Basic256 actually less in some cases. This can be explained by considering the algorithms included in the basic suites. The main difference between Basic128 and Basic256 is the use of AES128 and AES256 respectively, both of which are symmetric algorithms. However, both algorithm suites use the same asymmetric algorithm (RSA) which was the algorithm used in WS-Security to wrap the symmetric algorithms as they passed between web service and client. This explained the similar results. The lower impact for Basci256 when sending 1,000 books was unknown. It should be noted that in Figure 8 the results for ten books were not included due to the skewed results caused by a latency of 1 ms (which gave an invalid result when used as the divisor).

Figure 9 Latency due to each suite over a range of request sets

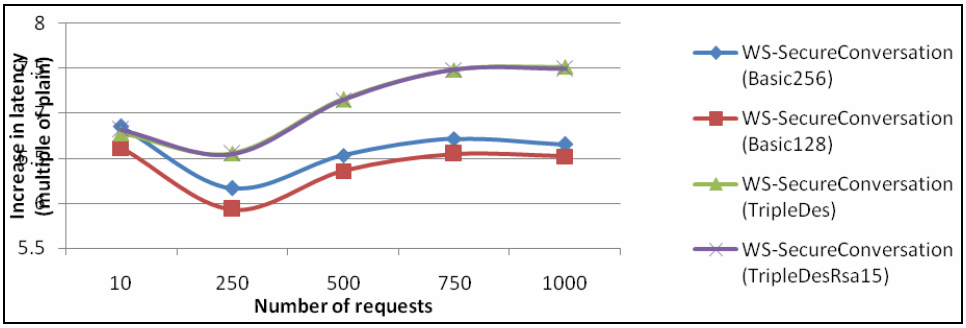


The second test in this section was designed to measure the performance impact of the different algorithm suites when using WS-SecureConversation to secure the SOAP message. In this case the message size was kept constant at 100 books. A small message size was chosen because WS-SecureConversation performs better with small messages. The different algorithm suites were tested against request sequences containing different numbers of requests (i.e., 10, 250, 500, 750 and 1,000). The total time was measured in this case, as opposed a single request/response in the previous test. Each sequence was run a number of times and the best result was recorded.

The results of this test are shown in Figures 9 and 10. The results were as expected in this test case with the Basic suites providing better performance than the TripleDes suites. When comparing Basic128 and Basic256, the response time was 20% longer with Basic256 than with Basic128. This was due to the use of WS-SecureConversation in the second test case which uses symmetric encryption after the initial handshake.

This meant that the lower processing needed for AES128 over AES256 was visible in this test. However, the difference between Basic128 and Basic256 was lower than expected.

Figure 10 Factor increase of each suite over a plain SOAP message



4.3 Analysis of WS-Security versus WS-SecureConversation

This test was designed to compare the performance impact of WS-SecureConversation against WS-Security. Encryption and signing were both included in the SOAP messages for both. All the messages were encrypted and signed using the Basic256 algorithm suite. A number of different request sequences were used and ranged from a large number of small messages to a small number of larger messages. These were:

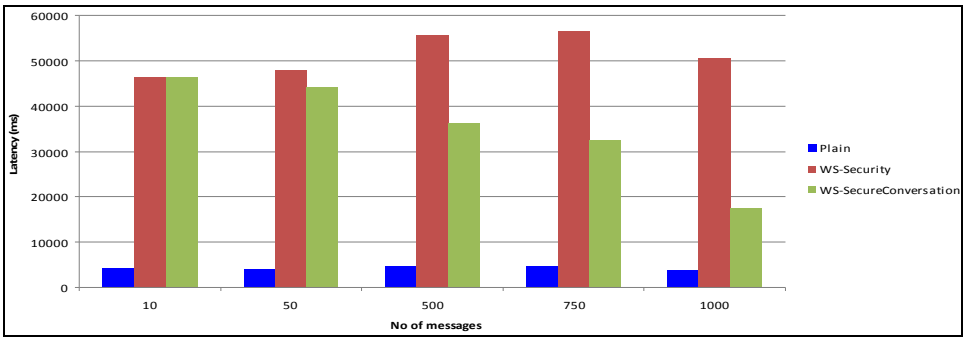
- 1,000 requests sent and 5 books in each response
- 750 requests sent and 100 books in each response
- 500 requests sent and 200 books in each response
- 50 requests sent and 3,000 books in each response
- 10 requests sent and 15,000 books in each response.

Each sequence was run multiple times with only the best time for each kept in the results. The chosen sequences tested different performance characteristics of web service stacks. The first determined how quickly web service stacks process messages with small amounts of data. The last sequence measured the performance when processing large amount of data. The security configurations tested were – No security; WS-Security (signing and encryption using the Basic256 algorithm suite) and WS-SecureConversation (signing and encryption using the Basic256 algorithm suite).

The expectation was that WS-SecureConversation would reduce the performance overhead associated with sending a number of secured messages and therefore reduce the latency for those messages. It was also expected that the improvement would be greater in the case where a large number of small messages was sent. The results in Figure 11 confirm the expected results. For the case of 1,000 small messages the performance due

to WS-SecureConversation was almost three times better than with WS-Security. However, as the number of requests decreased the performance improvement also decreased. This was also expected because of how WS-SecureConversation works. Before any messaging can take place between the client and the web service there must be a handshake where the symmetric key is shared. Although the use of the symmetric key is faster for signing and encryption the initial handshake adds additional time that is not needed when using WS-Security. When only a small number of requests were sent the benefit of using the symmetric keys was counteracted by the additional time needed for the handshake. This was shown in the final sequence where only ten messages were sent and there was almost no performance gain from WS-SecureConversation.

Figure 11 WS-Security vs. WS-SecureConversation



A second test of WS-SecureConversation was designed to measure the impact of the message size on the performance. In this test the size of the response message was changed but the number of requests remained constant. 500 requests were sent to the web service and the responses returned from the web service contained 50, 200, 400 or 600 books. Each sequence of requests was run with no security, WS-Security and WS-SecureConversation on the SOAP message. The results are shown in Figures 12 and 13.

Figure 12 Impact of message size on WS-SecureConversation

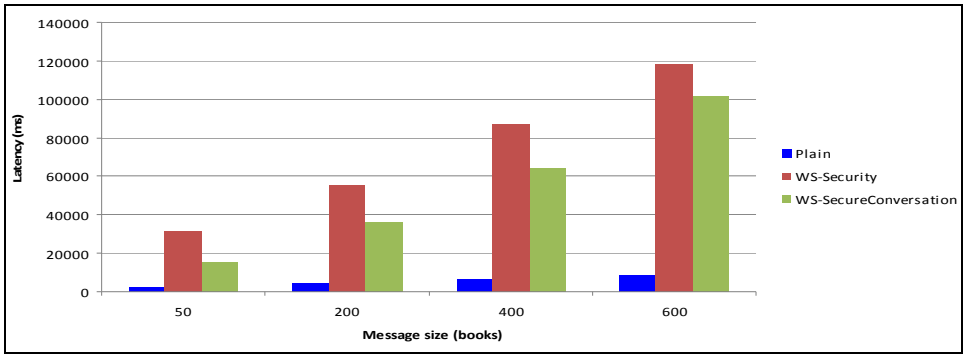
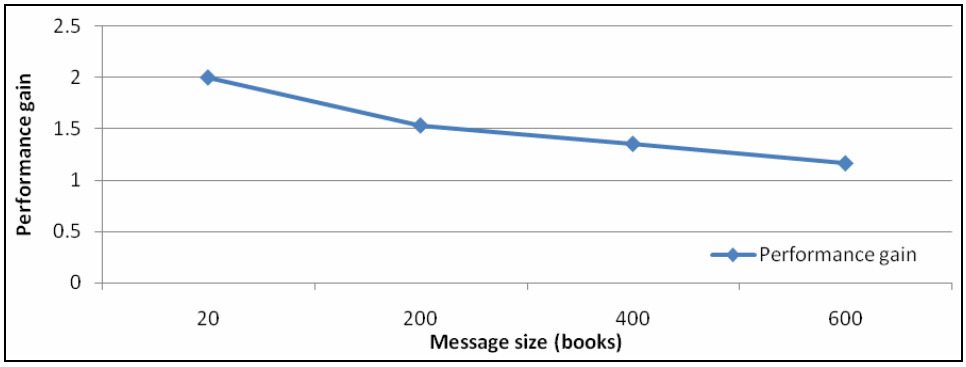


Figure 13 Factor of performance improvement of WS-SecureConversation over WS-Security (see online version for colours)



These results show a direct correlation between the size of the message and the performance gain achieved with WS-SecureConversation over WS-Security. This is irrespective of the number of messages sent. As the size of the response messages increased, the performance gain fell from a factor of 2 down to 1.2. This result can be explained by the processes used for the encryption and signing of XML messages. During the encryption stage it is encrypted using the chosen algorithm. Therefore the symmetric key used with WS-SecureConversation will perform better than the asymmetric key used with WS-security. However, signing an XML message involves a number of steps. The first is to convert the XML message into a canonical form. Once done the XML is digested to generate the hash value. This hash value is the value included in the actual signature. Generating the signature is the only step where either a symmetric or asymmetric key is used. As the message size increases the processing needed to canonicalise the XML message also increases but the time taken for generating the signature remains relatively constant. Therefore the increased canonicalisation limits the performance gain provided by the symmetric key during encryption and generation of the signature.

4.4 Impact of WS-Security when accessing a database in the web service

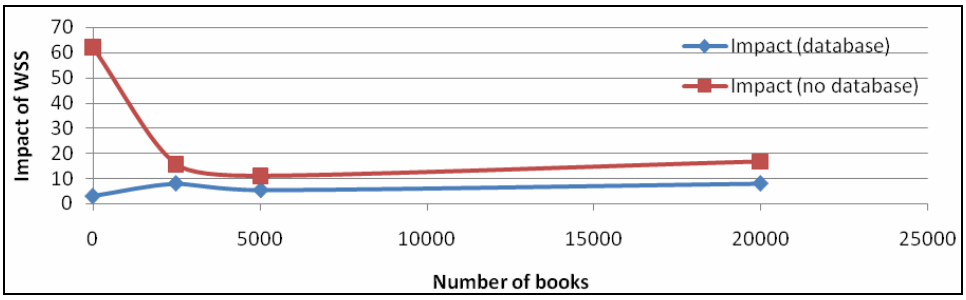
The inclusion of a database in the web service had a large impact on the WS-Security performance measurements due to the time spent accessing the database. This meant that any results measured when accessing a database did not give a true measure of the impact caused by the use of WS-Security to secure the SOAP messages. However, most enterprise applications use a database. Therefore, a simple test was prepared to determine the effects on WS-Security and WS-SecureConversation performance compared with no security when database processing was included. The MySQL 5.5 database was used in the web service and the persistence layer was implemented using Hibernate 3.3. Annotations were used to link the java objects to the database tables. X.509 certificates and the Basic256 algorithm suite were used for the encryption. The first test was designed to measure the impact of a database on the performance comparison between a SOAP message with no security and one that was signed and encrypted using WS-Security. A range of message sizes were chosen from very small to large (i.e., 1,

2501, 5094 and 20000). The same number of books was returned for the results retrieved from the DB and those generated in memory. For each sequence the best response time was selected and recorded (in ms). The results of these measurements are shown in Table 1. A comparison of the performance impact between plain and WSS messages (for both the database and non-database web service) can be seen in Figure 14. The vertical axis displays the factor increase due to WS-Security.

Table 1 Impact of DB on performance results

	1 book	*2493 books*	*5034 books*	*20000 books*
Database (plain)	109	124	296	1,109
Database (WS-Security)	93	985	1,593	8,832
No database (plain)	1	47	109	390
No database (WS-Security)	62	735	1,203	6,523

Figure 14 Performance impact due to WSS for database and non-database web service (see online version for colours)



As expected the results from this test show that the performance impact of using WS-Security to sign and encrypt SOAP messages was less when the additional database processing was included in the web service. This was due to the additional processing time of accessing the DB being included in the latency measurement irrespective of WSS. This had a larger impact on overall latency when no security was used but a smaller impact when WS-Security was used. Therefore, the overall performance impact was reduced. As can be seen from Figure 14 the performance impact was reduced from a factor of between 11 and 15 to a factor between 3 and 8. This was a substantial reduction in the impact of using WS-Security and would make its use far more favourable for enterprise applications. Note that the impact associated with a response message containing one book was ignored for the non-database case because the factor cannot be calculated accurately when the divisor is a value of 1. It should also be noted that these results would be valid regardless of the reason for the increased processing time in the web service. Any additional web service processing that increased the latency of a response from the web service would result in similar results. Therefore, the measurements for WSS impact represent more closely the actual performance impact of WSS. All the other WSS tests during this research have very little processing as part of the web service which explains the large difference in measured results in this case.

5 Conclusions

As expected the results proved that there is a large performance impact when using WS-Security. The measurements taken during this research demonstrated that the performance impact of using WS-Security for a simple web service was between 1,000 and 1,500%. This was higher than expected compared with other studies but could possibly be explained by the low processing required in the web service. As can be seen from the fourth set of tests the inclusion of the database processing in the web service resulted in an impact of between 300 and 800%. The use of the Basic256 suite may also have increased the impact but as can be seen from the results of test two this would possibly have been negligible. These results show that when dealing with a large volume of messages between a web service and a single client WS-SecureConversation provides greater performance than WS-Security. In the case of 1,000 small messages the impact of using WS-Security was a factor of 13. However, the impact of using WS-SecureConversation was only a factor of 4.5. Moreover, the performance gain over WS-Security will improve further as the number of messages increases. However, for a small number of messages the performance impact of WS-SecureConversation can be greater than that of WS-Security due to the additional cost associated with the initial handshake. The other interesting feature is the fact that the performance gain is more pronounced when signing and encrypting small messages. Therefore large performance improvements from WS-SecureConversation over WS-Security can only be achieved when messages are kept small and there are large numbers of messages expected.

When a database was queried as part of the web service the performance impact associated with WS-Security was reduced from a factor of between 11 and 15 to a factor of between 3 and 8. The reason for this was the latency impact of accessing the database, which remained relatively constant regardless of whether WS-Security was used. This reduced the factor difference between plain and WSS messages. An enterprise level web service would also have significantly increased processing requirements compared with the web service in this research. Therefore, although the use of WS-Security in an enterprise application would have an impact on performance, the effect would be less pronounced than the suggested by the results attained during this research.

As was shown in test two, the cost of using the Basic256 algorithm suite compared to the Basic128 suite is negligible when using WS-Security. Therefore, due to the increased security provided by the Basic256 suite there would seem to be no reason to use the Basic128 suite at all. This is not the case when using WS-SecureConversation where the trade off from the improved security is an increased performance overhead. In all cases the better performance and security provided by the basic suites would suggest that the TripleDes suites should not be used, unless there is an underlying reason which means they must be used.

References

Almeida, M. (2010) *Defacements Statistics 2008–2009–2010*, Zone-H.org [online] http://www.zone-h.org/news/id/4735 (accessed 27 May 2010).

Bartel, M., Boyer, J., Fox, B., LaMacchia, B. and Simon, E. (2008) *XML Signature Syntax and Processing*, 2nd ed. [online] http://www.w3.org/TR/xmldsig-core/ (accessed 8 January 2011).

Bhargavan, K., Corin, R., Fournet, C. and Gordon, A.D. (2007) 'Secure sessions for web services', *ACM Transactions on Information and System Security*, Vol. 10, No. 8, pp.36–52, doi:10.1145/1237500.1237504.

BouncyCastle (2011) [online] http://www.bouncycastle.org/ (accessed 8 January 201).

Boyer, J., Eastlake, D.E. and Reagle, J. (2002) *Exclusive XML Canonicalization*, Version 1.0 [online] http://www.w3.org/TR/xml-exc-c14n/ (accessed 21 February 2011).

Butek, R. (2005) 'Which style of WSDL should I use?'[online] http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/#N1021A (accessed 19 March 2011).

Chen, S., Zic, J., Tang, K. and Levy, D. (2007) 'Performance evaluation and modeling of web services security'. Paper appears in *IEEE International Conference on Web Services*, pp.431–438, doi:10.1109/ICWS.2007.139.

Christensen, E., Curbera, F., Meredith, G. and Weerawarana, S. (2001) *Web Services Description Language (WSDL) 1.1*, 15 March, W3C website [online] http://www.w3.org/TR/wsdl (accessed 2 November 2011).

Coarfa, C., Drushcel, P. and Wallach, D. (2006) 'Performance analysis of TLS web servers', *ACM Transactions on Computer Systems*, Vol. 24, No. 1, pp.39–69.

Davis, D. (2001) 'Defective sign and encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP and XML', *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pp.65–78.

Dierks, T. and Rescorla, E. (2008) *The Transport Layer Security (TLS) Protocol, Request for Comments: 5246* [online] http://www.ietf.org/rfc/rfc5246.txt (accessed 8 January 2011).

Discover Metro – Project Kenai (2010) [online] http://metro.java.net/discover/ (accessed 1 February 2011).

Domain Tools (2010) *Domain Counts & Internet Statistics* [online] http://www.domaintools.com/internet-statistics/ (accessed 1 February 2011)

Haas, H. (2003) *From SOAP/1.1 to SOAP version 1.2 in 9 points*, W3 [online] http://www.w3.org/2003/06/soap11-soap12.html (accessed 8 January 2011).

Ichikawa, T., Kasuya, T. and Matsui, M. (2000) 'Hardware evaluation of the AES finalists', *Proc. Third Advanced Encryption Standard Candidate Conference*, April, New York, USA, pp.24–32 [online] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.6049&rep=rep1&type=pdf.

Imamura, T., Dillaway, B. and Simon, E. (2002) *XML Encryption Syntax and Processing* [online] http://www.w3.org/TR/xmlenc-core/ (accessed 21 February 2011).

Jang, K., Han, S., Han, S., Moon, S. and Park, K. (2010) 'Accelerating SSL with GPUs [electronic version]', *ACM SIGCOMM Computer Communication Review*, October, Vol. 40, No. 4, pp.437–438.

Java Cryptography Extension (JCE) (2002) http://docs.oracle.com/javase/1.4.2/docs/guide/security/jce/JCERefGuide.html.

JSR 224: Java API for XML Based Web Services (JAX-WS) 2.0 (2007) Java Community Process website [online] http://jcp.org/en/jsr/detail?id=224 (accessed 3 January 2011).

Juric, M.B., Rozman, I., Brumen, B. Colnaric, M. and Hericko, M. (2006) 'Comparison of performance of web services, WS-Security, RMI, and RMI–SSL', *Journal of Systems and Software*, Vol. 79, No. 5, pp.689–700, doi:10.1016/j.jss.2005.08.006.

King, G., Bauer, C., Anderson, M.R., Bernard, E., Ebersole, S. and Ferentschik, H. (2011) *Hibernate Reference Documentation*, Hibernate website [online] http://docs.jboss.org/hibernate/orm/3.6/reference/en-US/pdf/hibernate_reference.pdf (accessed 17 March 2011).

Liu, H., Pallickara, S. and Fox, G. (2005) 'Performance of web services security', *Proceedings of 13th Annual Mardi Gras Conference – Frontiers of Grid Applications and Technologies*, 3–5 February 2005, Baton Rouge, Louisiana [online] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.66.3064&rep=rep1&type=pdf.

Metro Web Services (2010) *Metro Web Services Technologies at a Glance*, Oracle website [online] http://www.oracle.com/technetwork/java/index-jsp-137051.html (accessed 3 January 2011).

Moralis, A., Pouli, V., Grammatikou, S., Papavassiliou, S. and Maglaris, V. (2007) 'Performance comparison of web services security: Kerberos token profile against X.509 token profile', *ICNS '07 Proceedings of the Third International Conference on Networking and Services*, p.28, doi:10.1109/ICNS.2007.93.

Moralis, A., Pouli, V., Papavassiliou, S. and Maglaris, V. (2009) 'A Kerberos security architecture for web services based instrumentation grids', *Future Generation Computer Systems*, Vol. 25, No. 7, pp.804–818, doi:10.1016/j.future.2008.11.004.

Shirasuna, S., Slominski, A., Fang, L. and Gannon, D. (2004) 'Performance comparison of security mechanisms for grid services', *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pp.360–364, doi:10.1109/GRID.2004.50.

Sosnoski, D. (2010) 'Java web services: WS-SecureConversation performance', IBM Developers Works Series [online] http://www.ibm.com/developerworks/java/library/j-jws16/index.html, (accessed 1 March 2011).

Srirama, S.N., Jarke, M. and Prinz, W. (2007) 'A performance evaluation of mobile web services security', *3rd International Conference on Web Information Systems and Technologies*, pp.386–392, doi:arXiv:1007.3644v1.

Tang, K., Chen, S., Levy, D., Zic, J. and Bo, Y. (2006) 'A performance evaluation of web services security', *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference*, October, pp.67–74, doi:10.1109/EDOC.2006.12.

Van Engelen, R.A. and Zhang, W. (2008) 'An overview and evaluation of web services security performance optimizations', *Proceedings of the 2008 IEEE International Conference on Web Services*, pp.137–144, doi:10.1109/ICWS.2008.102.

Verisign (2010) *Verisign 8-K Current Report* [online] https://investor.verisign.com/secfiling.cfm?filingID=1193125-10-213453 (accessed 1 February 2011).

WS-I (2010) *WS-I Basic Security Profile Version 1.1*. [online] http://www.ws-i.org/deliverables/workinggroup.aspx?wg=basicsecurity (accessed 8 January 2011).

WSS (2006) *Web Services Security: SOAP Message Security 1.1* [online] http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf (accessed 1 March 2011).

WS-SecureConversation 1.4 (2009) [online] http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/ws-secureconversation.html (accessed 8 January 2011).

WS-SecurityPolicy 1.3 (2009) OASIS website [online] http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.3/os/ws-securitypolicy-1.3-spec-os.pdf (accessed 5 January 2011).