How Far Web Services Tools Support OASIS Message Security Standards?

Shambhu Maharaj Sastry Sistla
How Far Web Services tools support OASIS Message Security Standards?

Submitted by Shambhu Maharaj Sastry Sistla to the University of Skövde as a dissertation towards the degree of M.Sc. by examination and dissertation in the School of Humanities and Informatics.

December 7, 2005

I hereby certify that all material in this dissertation which is not my own work has been identified and that no work is included for which a degree has already been conferred on me.

Signature: ______________________________________

ii
How Far Web Services tools support OASIS Message Security Standards?

Abstract

There is a great deal of interest burgeoning in the intellectual community regarding Web Services and their usage. Many writers have tried to bring awareness about some unconceived threats lurking behind the enticing Web Services. Threats due to Web Services are on an all time high giving an alarming knock to the Web Services security community. This led to the, Organization for the Advancement of Structured Information Standards (OASIS) made some constraints mandatory in order to standardize message security and these constraints and specifications are presented through a document called WS Security -2004. This work is an attempt to check the support offered by various Web Services Tools available currently. It introduces the reader to Web Services and presents an overview of how far some of the tools have reached in order to make the Web Services environment safe, secure and robust to meet the current day’s requirements. A quantitative approach was taken to investigate the support offered by servers like BEA, Apache Axis etc. The conclusions drawn show that most of the tools meet the imposed standards but a lot more is expected from the web community and these tools; if at all the visions about safe and secure Web Services are to be realized.
Table of Contents

1 Introduction ............................................................................................................. 1
  1.1 Service Oriented Architecture (SOA) ................................................................. 1
  1.2 Web Services ....................................................................................................... 2
  1.3 Problem ............................................................................................................... 2
  1.4 Motivation for the Problem .................................................................................. 3
  1.5 Aim ....................................................................................................................... 3
  1.6 Objectives ............................................................................................................ 3
  1.7 Research Study .................................................................................................... 4
  1.8 Delimitations ....................................................................................................... 5
  1.9 Expected Result ................................................................................................... 5
  1.10 Expected Contribution ....................................................................................... 5
  1.11 Summary ............................................................................................................ 6

2 Background ............................................................................................................. 7
  2.1 Historic overview ............................................................................................... 7
  2.2 What Technology enables Web services? ............................................................. 10
    2.2.1 XML and XML schema: It’s role in web services ........................................ 10
    2.2.2 Simple Object Access Protocol (SOAP) ....................................................... 11
    2.2.3 SOAP Physical Structure ........................................................................... 12
    2.2.4 WSDL ......................................................................................................... 12
    2.2.5 UDDI ......................................................................................................... 13
  2.3 Security in Web Services ..................................................................................... 13
  2.4 Attempts made to combat the concerns in Web Services ..................................... 14
    2.4.1 Confidentiality ............................................................................................ 14
    2.4.2 Integrity ....................................................................................................... 16
    2.4.3 Identity ....................................................................................................... 17
    2.4.4 Non-Reputation .......................................................................................... 18
  2.5 Basic Mechanisms to Handle Message Security ................................................ 18
    2.5.1 The role of SSI ............................................................................................ 18
    2.5.2 Concerns Raised On SSL .......................................................................... 19
  2.6 Summary ............................................................................................................. 20

3 What is WSS and what is its Contribution? ............................................................. 22
  3.1 Potential Reasons for releasing WSS-2004 ......................................................... 22
  3.2 Web Services Message Specification standard-2004 ......................................... 25
    3.2.1 WSS: Old Actors in New Key-Roles ........................................................... 25
    3.2.2 The Elements of WSS- 2004 ..................................................................... 27

4 Achieving the Objectives ......................................................................................... 32

iv
4.1 Objective 1: Understand WS-Security 2004 ..............................................32
4.2 Objective 2: Selection of Web Services tools ...........................................32
4.3 Objective 3: Determination of rating procedure ........................................32
4.4 Objective 4: Analysis of Results .............................................................33

5 Research Study ............................................................................................34
  5.1 To determine how should the rating be done .............................................34
    5.1.1 Namespaces Mandatory Issues .........................................................34
    5.1.2 ID Reference Mandatory Issues .....................................................35
    5.1.3 Security Header Mandatory Issues ..................................................35
    5.1.4 Security Tokens Mandatory Issues ...............................................35
    5.1.5 Token References Mandatory Issues ..............................................35
    5.1.6 Signatures Mandatory Issues .........................................................36
    5.1.7 Encryption Mandatory Issues .......................................................36
    5.1.8 Security Time Stamps Mandatory Issues ........................................37
  5.2 To Get the SOAP Message Content ........................................................37
  5.3 Check the demanded constraints and present the results ..........................37

6 Results, Evaluation and Analysis ................................................................40
  6.1 Results and Evaluation ...........................................................................40
  6.2 Analysis ...................................................................................................41

7 Conclusion .....................................................................................................43
  7.1 Summary ..................................................................................................43
  7.2 Contribution .............................................................................................43
  7.3 Future Work .............................................................................................44

References ..........................................................................................................45

Appendix A: Rating Reasons .............................................................................48
Appendix B: Listings ..........................................................................................55
Appendix C: Codes ............................................................................................64
Appendix D: System Information .......................................................................74
1 Introduction

1.1 Service Oriented Architecture (SOA)

Endurance of any concrete thing no matter which ever example is considered, rests on the strength assured by it. A strong chassis is the requirement for a strong component, right from a small linchpin in an enterprise to an enterprise wide application. Chassis here refers to the underlying model or the architecture of the any application. Tsalgatidou and Pilioura [34], illustratively explain this paradigm. They cover everything right from its need to its benefits and an account of which is presented below.

The working model of Service Oriented Architecture (SOA) is shown in Figure 1; the working is explained through the Web Services perspective as it is easy to understand and further helps the reader to asses the coming topics. The Service Oriented Architecture may be defined as a loosely coupled architecture that rests on the paradigm of request response model. The real essence of the architecture lies on the paradigm of platform neutrality and abstracting underlying implementation details from the service provided. The central working model primarily rests on the roles described or involved in the model. There are three primary roles as per Tsalgatidou and Pilioura [34]

![Diagram of Service Oriented Architecture](image)

**Service provider:** A service provider on the top, in Figure 1 is the party that provides software applications for specific needs as services. Service providers publish and update their services so that they are available on the Internet. From a business perspective, this is the owner of the service. From an architectural perspective, this is the platform that holds the implementation of the service.

**Service requester:** A requester is the party that has a need that can be fulfilled by a service available on the Internet. From a business perspective, this is the business that requires certain function to be fulfilled. From an architectural perspective, this is the application that is looking for and invoking a service. A requester could be a human user accessing the service through a desktop or a wireless browser; it could be an application program; or it could be another Web Service. A requester finds the required services via a service broker and binds to services via the service provider.
**Service broker** As it can be seen from the figure this role acts as a bridge between the service provider and the service requester. This party provides a searchable repository of service descriptions where service providers publish their services and service requesters find services and obtain binding information for these services. The reader must note that there is no business perspective or architectural perspective in this role as service broker has one primary motive to find the requested service provider irrespective of the perspective.

The service provider publishes, or updates his services at a broker. The service seeker finds the service provider via the broker and establishes contact with the service provider. The provider finally requests the service provider for services and when service is invoked, binds to the services he needed. This explains the entire working in short.

### 1.2 Web Services

W3C defines Web Services as a “Software system designed to support interoperable machine-to-machine interaction over a network”. Web Services can also be thought of as integration of the five technologies namely HTTP, XML, SOAP, WSDL (Web Services Description Language) and UDDI (Universal Description Discovery and Integration Language). In the previous section the reader is introduced to SOA and the technical aspects behind the SOA. All these technologies act in tandem to provide interoperable machine-to-machine interaction.

The actual working of the Web Services includes the involvement of the intermediate websites as the paradigm works on the concept of entity-to-entity connectivity and not peer-to-peer connectivity as per the Web Services definition by W3C. These web sites act as hops before the web service is completely provided (refer to Figure 7 and subsection 2.5 for a better understanding). These intermediate websites are sometimes termed as intermediaries or middle men who are vulnerable to security threats like replay attacks etc. Refer to Cole et al. [6], King [18], Burns [3] (or Section 3.1) for further information on topics regarding threats. Cole et al. (2005) [6] alone cites nearly more than 22 known types of attacks which disrupt the Web Services. These begin with viruses, denial-of-service (DoS) attacks, Back door, Spoofing, Man-in-middle, Replay, TCP/Hijacking, Fragmentation, Weak-Keys, Port scanning, Dampster Diving, Birthday attacks, eavesdropping, war driving etc... and describing each of these cannot be done in this article but an overview of this area would be presented shortly.

This led to the need for a strict Web Services security. The IETF (Internet Engineering Task Force) and other technical and research communities quickly responded to this need by coming up with solutions like encryption, signatures, security tokens and other schemes and technologies like SSL/TSL etc. but each of these suffered a set back due to issues such as proprietary or lack of uniform convention of these technologies regarding implementation. The reader would be shortly exposed to these in coming sections. Chapter 3 explains these topics in detail. Thus WS Security committee released OASIS Message Security Specification in March 2004 [23] (from here on WS Security – 2004 or WSS-2004 is used in the text to refer to the OASIS Message Security Specification) to combat such issues.

### 1.3 Problem

Due to the increasing threats in the Web Services, many developers and researchers are doubtful of the assistance that Web Services tools (the software which helps in implementing Web Services is referred to as Web Services tool in this article) provide in implementing

---

1 Interested readers are requested to kindly refer to chapter 17 of the text book Network Security Bible [Error! Reference source not found.] for further information on the topic.
security standards in Web Services, Newcomer and Lomow [21]. This type of doubts is the problem considered in our article. It should be clearly noted that although the same problem is cited in different aspects by Newcomer and Lomow [21], O’Neill [24] and Yang [39] in different research needs, this article does not try to address the impacts of the outcomes of the work but simply cites the expected possible beneficiaries in the 1.10 subsection.

1.4 Motivation for the Problem

The Westbridge technologies [37] brings forth the unaddressed issues in Web Services security such as, “Difficult to prove that a given security policy has been implemented”, Do the currently available Web Services tools really support the WSS-2004 security standards is a question that is to be answered. Many software giants have done a work similar to this but it is a practical survey of their products performance and does not often address the issue from the WSS-2004 perspective as much as they address the security perspective. Some companies like BEA have developed applications to test the performance of their servers and the surveys are done on the feasibility aspects and cannot be considered to be absolutely objective as they were designed and performed in a closed community and cannot be always trusted.2

There is also motivation from few points raised by Newcomer and Lomow3 [21]: Kearney et al. [16] regarding adoption of standards by the commercial web tools. These doubts add to the keenness to verify the constraints specified in WSS-2004 in the selected tools. The doubts that authors express are due to the SOAP Message that is being used in Web Services. WSS-2004 is based on SOAP messages. These SOAP messages are known to bypass firewalls and hence the author has suspicions over the adoption of WSS-2004 in Web Services tools. If the verification of tools is done the fate of author’s fears would be exposed.

1.5 Aim

As presented above, there is a suspicion about the Web Services tools following the security constraints specified in the WSS-2004. The aim of this work is to verify whether different Web Services development tools follow the security constraints specified in WSS-2004. The proposed work begins with the selection of web service tools for evaluation then grouping the constraints, implementing a sample from the tutorials available on the internet, gathering the SOAP message generated, verifying the constraints in it, and finally try to account for the conclusions reached. The objectives of this work are presented below in Section 1.6 and the chapter 4 discusses how these were established. Kindly refer to the respective sections for further details.

1.6 Objectives

There are four objectives of this work. These are:


---

2 http://www.webservices.org/index.php/ws/content/view/full/61096 this link tells us that senior technician architect, Hal Lockhart of BEA Systems, also the lead of the OASIS InterOp team assuring its completely support to WS-Security.

3 Refer to chapter 8 “Core Concepts”
2. **Selection of Web Services tools**: To reason why only certain tools were chosen for evaluation from all the currently available Web Services tools.

3. **Determination of rating procedure**: To formulate codes (cases) from constraints specified in WSS-2004, which can be used as token bearers to hold the rating during analysis phase.

4. **Analysis of observations**: To check for the constraints demanded and rate them with reasons, tabulate the ratings. Finally analyze and discuss the results.

### 1.7 Research Study

The flow chart in the Figure 2 shows the overview of the research approach followed.

![Flow chart of the research approach adopted](image)

To fulfill the aim, a quantitative analysis is adopted i.e., a scientific approach of analyzing the issue through experimentation and deriving conclusions by analyzing the results of the experiments (Berndtsson et al.) [1]. WS-Security 2004 draft contains the list of constraints on message security. The actual plan and plot of the study that is expected to be done is presented in this section.

After selecting Web Services tools (kindly note that the reasons for selecting tools are explained in detail in accomplishment of second objective), a brief list of the things to be evaluated is to be made from the WS-Security 2004 draft after understanding the concepts and constraints mentioned in it. Codes (like Must-Signatures, Must-Encryption etc) are assigned to the issues pertaining common element (i.e. all the MUST constraints in signature domain are assigned Must-Signature code etc) from the listing made to assist in carrying evaluation. It should be realized that same format of study cannot be followed on all the selected tools because some of the tools are open software and some of them are proprietary which provide...
limited access to the required details. So, as per the tool, analysis is to be carried out. The reader must note that although the format followed is different, the ultimate requirement is to extract the SOAP content. As the generated SOAP content is common for each Web Services tool the SOAP content acts as common means to perform the comparison. The next objective would be to gain access to the SOAP message content created by the tool used for evaluation. Once the SOAP content is gathered it is checked for the constraints specified in the WS-Security 2004 document and as per the perceived notions and observations it is to be done along with the reasons for a particular rating.

1.8 Delimitations

The reader must note that some constraints require knowledge of the tool’s architecture like XML schema followed, parser used etc. such issues are available in some cases like Apache Axis which is an open software and in some cases it is not available like in case of Microsoft and BEA. The verification in such cases is ignored. A brief reasoning for every aspect of the conducted work is presented in the respective contexts and the reader could find these in the coming sections. A significant delimitation that impedes this work is the lack of a significant distributed network (infrastructure) but it is expected that most of the evaluation was being successfully carried except two points in two issues. Those details are also provided in the Section 6.2 which is concerned with situations of multiple hops.

1.9 Expected Result

The most common and obvious expected result is the summarization of the support for WS Security 2004 from the Web Services tools side. That is how many? And which of the specifications of the WS Security 2004 are being supported by the tools used for verification. The reader is biased to misunderstand the word, “support” repeatedly being used in the context. In the current context this word means that, “if the SOAP message generated from the tool is pertaining to the limitations and standards specified in the WS Security - 2004 document” then the tool is said to support the standards. As per the observations made and notions conceived on conclusion of evaluation and verification, a rating is given to every point that is verified.

1.10 Expected Contribution

This work may help the researchers in determining the tool’s adoption or rejection by the Web Services developing community for their regular usage. It can even help the researchers who are working on the arguments extended by the authors like Newcomer and Lomow⁴ [21] regarding adoption of new security trends in current Web Services scenario. Our work can help one to choose a tool that suits his or her requirement as the analysis would also provide reasoning for the rating.

The results can also be helpful to the researchers who are investigating issues raised by O’Neill [24] and Yang⁵ [39] who express concerns regarding SOAP messages in Web Services. They refer to bypassing firewalls, as a key point which would hinder the adoption of Oasis Security Draft [23] by the web tools.

⁴ Refer to chapter 8 “Core Concepts” and “challenges to adoption”
⁵ Refer to page number 20 of the volume, right hand side first paragraph.
1.11 Summary

This subsection presents an overview of how the things are arranged in this document. Chapter 2 explains Web Services in a broader perspective. Section 2.1 presents an overview of existing methods and why they cannot be used with new Web Services requirements. Section 2.2 explains the technical aspects that make Web Services a reality. Section 2.3 addresses the security in Web Services; this Section lists the concerns looming in the region of Web Services. Section 2.4 tries to present an overview of steps taken to combat the lurking threats in Web Services scenario. The subsections in this section expose the primary aspects considered in Web Services security domain like confidentiality, integrity authentication etc. and what is being done so far to achieve these aspects. Subsection 0 presents an overview of TSL and its liabilities to meet the current Web Services demands. Chapter 3 introduces the WS Security – 2004 document to the reader. Every subsection of Section 0 presents the new roles of the existing technologies like signatures etc. Chapter 4 discusses how the objectives set were established. Each section of this chapter discusses one of the objectives set. Chapter 5 discusses the research study done. Core work of the current document deals with the explanation of the evaluation work that is carried out. It explains how codes used for rating are formed and presents a brief description of the test bench used for conducting the work. Chapter 6 presents the evaluations of the gathered codes. Chapter 7 presents the conclusions reached in the current work. Subsection 7.1 presents the short summary of the current work with subsections 7.2 and 7.3 illustrating the contributions and future works.
2 Background

After a brief introduction of the SOA and its working in Section 1.1 it is now time to consider the strength in existing mechanisms like CORBA and DCOM to support usage of SOA.

2.1 Historic overview

A significant issue on SOA would be the comment on its Simplicity from Chappell and Jewell\(^6\) [4]. Bloomberg\(^1\) [2] also expresses that SOA is not a fresh bud in the Web Services scenario but feels that the existing approaches to service orientation, however, suffered with few problems. First, they were tightly coupled, which meant that both ends of each distributed computing link had to agree on the details of the API. Secondly, such Service-oriented architectures were proprietary. Microsoft unabashedly controlled DCOM, and while CORBA was ostensibly a standards based effort. In practice, implementing CORBA architecture typically necessitated the decision to work with a single vendor’s implementation of the specification, because each vendor’s interpretation of the standard varied enough to prevent seamless interoperability. Finally, CORBA and DCOM were fine grained, which meant that service requests and responses typically contained small amounts of specific information, requiring many round trips between the consumer and the producer of the service. In spite of all these issues, the concept of service orientation continued to make sense, provided that the problems of proprietary approaches, tight coupling, and fine granularity can be solved. It is within this architectural context that Web Services were first imagined. Service-oriented architectures implemented with Web Services are a significant improvement upon DCOM and CORBA’s weaknesses. The basic reason behind the adoption of SOA for the Web Services is nothing but the simplicity of design and more than that it is loosely coupled.

Rosenberg and Remy\(^2\) [28] expressed the same views regarding DCOM they feel that DCOM though has transport neutrality it suffers from high complexity and is moreover very fine grained. Regarding CORBA, they express fears about complexity which makes less scalable than Distributed Computing Environment (DCE). One thing the reader must approve is the fact that CORBA is highly object oriented and is vendor specific (i.e. IONA etc)\(^3\). Coming to Web Services, the issues highlighted above are dealt here. Tsalagidou and Pilioura [34] elucidate the most prime issues and reasons, as they rate them as the secrets of success behind SOA’s adoption for Web Services. They express that Web Services through the underlying SOA design facilitate the following:

**Easy and fast deployment:**

Now one can feel the absence of investment and time delays in Web Services as there is flexibility factor added to the paradigm and moreover there is no reason for reengineering of legacy systems etc. One can develop new Web Services by reusing and/or combining existing ones. For example the portal, Web Service provides a set of high-level features by maintaining

---

\(^6\) Refer to “Chapter 3. SOAP: The Cornerstone of Interoperability” in the text book java Web Services book
\(^1\) Refer page number 5 Loose coupling’s role under …
\(^2\) Refer to chapter2 the paragraph just above “The Web: The Global Network for Information Exchange”.
\(^3\) Though CORBA is said have very strong on Security many scientific skeptics are doubtful about its performance in the situation of multiple hops. Refer to Weerawarana et al. (2005) [36] for further information. And this is included in SOAP description provided later [Messages,] The problem arises due to the fact that Messages can be routed based on the content of the headers and the data inside the message body in soap and it is not possible in CORBA and DCOM
proportion over the lower level Web Services for portfolio management, stock quote and others.

**Interoperability:**
Due to the XML based **Interface Definition Language** (IDL) and SOAP message protocol involving Request-Response architecture, Web Services can freely interact amongst them without user or any other role involvement. What this enables is it provides the developers to be independent of development domains of the producer or the consumer. Thus, the net output is easy and **interoperable** environments for developers to work with **reduced time consumption** and **cost** as by-products.

**Just-in-time integration:**
Conventional systems severely suffered from what is termed as “Brittle-Coupling” i.e. high association between the components involved in the architecture. This in turn rendered the systems passive even to the slightest change or modification that would be done in order to meet the latest changes required. Now with the introduction of the new concepts the systems facilitate smoother decoupling thereby enabling **just-in-time integration** of the applications. An admired output which is discovered is the ability of the applications to discover and balance network options for more scalable and better performance of message transfer. It is up to the application to select the needed network settings through self configuring, adaptive and robust mechanisms to achieve better performance. Now that there are less single point failures **real time aspects** like **scalability**, **fault tolerance and reliability** scale well when overall application scenario is considered.

At this point the reader is expected to understand the need for new type of technical architecture for Web Services rather than using the existing technologies such as DCOM, CORBA, IMS etc. There can be many more reasons in the literature but that is not what this work aims to present and even these are discussed here to bring the very best points cited in the literature under one umbrella. This makes it easy for the reader to understand the need for new proposals in Web Services. The scientific community expects that the new standards brought forth through the WSS-2004 would **redeem** the issues of fiasco in keeping up to the needs and goals expected of the Web Services.

There is a short final comparative summarization done for the readers to assess the severity and depth of the problem. Kindly note that this table is not taken from any where and it is also a part of analysis done during targeted work, the references made to build up the table are in [] brackets. This work of building a summarization table is also a contribution to the research community from the current work. It should also be noted that the comparison analysis done here is not the primary concern of the intended work. The analysis is done only to attribute a comprehensive overview of the introductory study. The ending up of the analysis as a significant contribution is a mere serendipity and is not something initially intended. That is the primary reason behind the table of comparison not being focused in discussion. Reader(s) if interested can refer to the reference numbers in [] for more information.
**Table 1: A small Comparative study between these distributed Computing packages.**

<table>
<thead>
<tr>
<th>Areas</th>
<th>DCOM</th>
<th>CORBA</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coupling[26]</td>
<td>Tightly Coupled (depend particular execution environment)</td>
<td>Tightly Coupled (depend particular execution environment)</td>
<td>Loosely Coupled Tightly Coupled (independent of execution environment)</td>
</tr>
<tr>
<td>Semantic level[17]</td>
<td>CORBA is strongly based on C data structures—C, C++, and Smalltalk worked</td>
<td>SOA is language-agnostic</td>
<td></td>
</tr>
<tr>
<td>Business processes?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Proprietary[27]</td>
<td>Yes (Microsoft)</td>
<td>Yes (Sun)</td>
<td>No, Open</td>
</tr>
<tr>
<td>Primary Motive[27]</td>
<td>Effective Fault tolerance, Load Balancing etc. [27]</td>
<td>Secured Distributed Peer-to-Peer Transport.</td>
<td>Loose coupling And flexibility for service orientated concepts.</td>
</tr>
<tr>
<td>Bandwidth[26]</td>
<td>Acceptable though ineffective.</td>
<td>Ineffective(Peer to Peer)</td>
<td>Effective (Any entity to any entity)</td>
</tr>
<tr>
<td>Interface Definition Language (IDL)[21,12]</td>
<td>Not Needed Aggregation supported directly through Interface IDs (IID).</td>
<td>impossible to use the interoperability transport without the interface definition language (IDL)</td>
<td>SOAP can be used without WSDL</td>
</tr>
<tr>
<td>Service Discover[21]</td>
<td>SSDP Discovery Service and Remote Registry Service</td>
<td>CORBA naming/trading service</td>
<td>UDDI</td>
</tr>
<tr>
<td>Adoption[23]</td>
<td>Difficult to Adapt</td>
<td>Difficult to Adapt</td>
<td>Easy To Adapt</td>
</tr>
<tr>
<td>Cost Associated[21]</td>
<td>Low cost of Adoption</td>
<td>Low cost of Adoption</td>
<td>More cost adoption, involves reengineering</td>
</tr>
<tr>
<td>Integration[24]</td>
<td>Very sensitive to changes hence <em>Just-in-time integration not easily feasible</em></td>
<td>Very sensitive to changes hence <em>Just-in-time integration not easily feasible</em></td>
<td>Very flexible to changes hence <em>Just-in-time integration possible</em></td>
</tr>
<tr>
<td>Grained[26] (Level of Detail)</td>
<td>Finely Grained (i.e. contain specific information, like number of round trips)</td>
<td>Finely Grained</td>
<td>Coarsely Grained (XML’s document-oriented structure enables systems to exchange coarse-grained messages)</td>
</tr>
</tbody>
</table>
2.2 What Technology enables Web services?

Ort [25], Rosenberg and Remy [28], and Tsalgotidou and Pilioura [34], present a brief description of the working model of Service Oriented Architecture in Web Services. Gokhale et al. [11] add more to the description given by above authors by making a comparative study between Web Services and CORBA’s working models. A serious attempt is made in our work to address this issue. The text written below consists of excerpts taken from the work of the authors cited above and a reference to it could be made for a deeper insight in the topic.

![Diagram](image)

Figure 3: Step wise operation of service oriented architecture in Web Services (Tsalgotidou and Pilioura) [34]

The scenario shown in the Figure 3 begins with the Service Provider. When a service provider launches a new service he registers the service with the UDDI Registry. The UDDI registry, in the second step confirms the publishing of the service. At development time, the registry can be searched (Figure 3, steps 3 and 4) by a programmer for suitable services and can be used to locate the appropriate Web Service Definition Language (WSDL) file (Figure 3, steps 5 and 6). A WSDL document may refer other WSDL documents and XML Schema (XSD) documents that describe data types used by Web Services. The WSDL document is stored on the service provider. Alternatively, it may be stored in special XML repositories; in this case, better performance is achieved as the network overhead for the service provider is reduced, since in the latter case (i.e., storing WSDL document in XML repositories) hosts only execute the service requested. The service can be a Servlet, a CGI script, a Perl script, etc. After the programmer has studied the specifications for the Web Service described in the retrieved WSDL documents, s/he generates client proxies so that the application can access the service (figure 2, steps 7 and 8). Alternatively, development tools can be used to generate the required client proxies.

2.2.1 XML and XML schema: It’s role in web services

XML (eXtensible Markup Language) is a markup language for documents containing structured information. Structured information contains both contents (words, pictures, etc.) and some indication of what role that content plays (Walsh [35]). Web Services use XML to
describe their interfaces and to encode their messages. XML Schema is a way of describing
the rules for a particular XML instance (for a more formal definition refer to W3C standard
definition on web or [35] numbered reference in references). XML is the foundation for the
Web Services and all the standards for describing, discovering and invoking Web Services are
based on XML. Core security standards like XML Encryption, Security Assertion Markup
Language (SAML), and WS-Security are based on XML and XML-Schema. XML documents
are text based documents and hence tools to parse them are needed only when document is of
large size. The format of tagged elements arranged in hierarchical order makes XML
documents easy to comprehend.

Web Services use XML to describe their interfaces and to encode their messages. As per
Ort [25]; Tsalgatioud and Pilioura [34] XML-based Web Services communicate over standard
Web protocols using XML interfaces and XML messages, which any application can interpret.
But XML by itself does not ensure effortless communication. Verifying data integrity is very
challenging with XML due to the differences between the platforms and XML parsers yielding
logically equivalent documents. The applications need standard formats and protocols that
allow them to properly interpret the XML. Hence three XML-based technologies emerged out
as standards for Web Services.

1. Simple Object Access Protocol (SOAP) defines a standard communications protocol
   for Web Services.
2. Web Services Description Language (WSDL) defines a standard mechanism to
   describe a Web Service.
3. Universal Description, Discovery and Integration (UDDI) provide a standard
   mechanism to register and discover Web Services.

2.2.2 Simple Object Access Protocol (SOAP)
Rosenberg and Remy [28] give a thorough picture of the SOAP architecture. An introduction
about SOAP is presented here. SOAP is supposed to address two primary issues. Primarily,
the issue of achieving satisfactory model for business processes which CORBA and other
technologies fail (referring to fourth point in the table) and secondly achieving peer-to-peer
functionality as the conventional processes were able to. Then the question is, how can these
two be achieved in parallel? The answer to this question lies in the SOAP implementation
mechanism. SOAP can be said to be working in two modes: Text Based Document Format
and RPC/Client Model.

2.2.2.1 Document/Literal mode operation
This is the mode of the operation used when business processes are concerned. The reasons
behind this are, primarily SOAP uses HTTP GETS and POST methods for the message
transfer. It works as an extension of HTTP which makes this possible. When the forms are
submitted the web server is expected to understand these submissions. Secondly, effective
bandwidth utilization is achieved in low bandwidth scenarios through this mode of operation.
Reliability is also a by-product in this mode and a real time issue which severely hinders the
architecture of work if not checked, as denial of service attacks, non-repudiation etc.( will be
introduced shortly under security concerns) may crop up.
2.2.2.2 RPC Encoded mode
The features of this mode can be summarized as; this is a mode which deals with the remote procedures and transparent calls to the clients and other coarse grained work. It deals with operations involving synchronous mode of transmission. In this mode it captures programming language data elements such as data types of variables, etc in a language neutral and interoperable format. Once these details are captured then one can foresee how a remote procedure call can be expressed in XML. The data details are stored as child elements in XML elements along with the remote procedure names. Thus later when needed these are invoked via the XML file.

2.2.3 SOAP Physical Structure
The protocol uses an envelope which wraps the security content and the message content into a unique case. This structure acts like a case to hold the message and security content. Each envelope has the following:

2.2.3.1 Header
This part is the hub of the activity as, it is the place where:
1. Security information is stored.
2. System level information used to establishing connection is held.
3. Information about transactions, routing, payment, delivery guarantee, etc is stored as a part of the header information. In conventional techniques these are coded once and for all in the data structures used and there are no dynamic alterations except to recode them.

2.2.3.2 SOAP body
This contains the following things,
1. Payload of the message.
2. XML document, for application, initiating the SOAP message.

Before concluding this introduction about SOAP it is expected that the reader notices the significance of XML-RPC over document/literal based. It is crucial for building a distributed application as it forms the backbone of the architecture.

Weerawarana et al. [36] elucidate the mechanism behind routing as they write that; (reference to foot note 9) messages can be routed based on the content of the headers and the data inside the message body. You can use tools developed for the XML data model to inspect and construct complete messages. Note that such benefits were not available in architectures such as DCOM, CORBA, and Java Remote Method Invocation (RMI), where protocol headers were infrastructural details that were opaque to the application.

The material presented in this section is a very fine digest of the text extracted from the references like Weerawarana et al [36], Newcomer and Lomow [21], Tidwell et al. [33], Rosenberg and Remy [28] and many more. The reasoning behind the issues raised might not be explicitly available to the readers directly from the references and require some analysis on the part of reader to acknowledge the issues.

2.2.4 WSDL
Systinet [30] describes WSDL as an XML vocabulary for describing a Web Service. A WSDL document describes the functionality of a Web Service, the way it communicates, and its access. WSDL provides a structured mechanism to describe the operations, a Web Service can
perform the formats of the Messages that it can process, the protocols that it supports, and the access point of an instance of the Web Service. A more formal and detailed explanation is also presented in Weerawarana et al. [36] as they view WSDL as an XML format for describing (network) services as a set of endpoints that operate on messages containing either document-oriented or procedure-oriented information. It is even felt that it is the most mature form of metadata describing Web Services as it allows the developers to present “Functional” characteristics of a Web service, i.e. what actions or functions the service performs in terms of the messages it receives and sends. It is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate.

Weerawarana et al. [36] believe that WSDL offers a standard, language-agnostic view of services it offers to clients. It also provides compatibility with existing applications and services and allows interoperability across the various programming paradigms, including CORBA, J2EE, and .NET. A WSDL document has two parts: abstract definitions and concrete descriptions. The abstract section defines SOAP messages in a language- and platform-independent manner. In contrast, the concrete descriptions define site-specific matters such as serialization.

WSDL provides support for a range of message interaction patterns. It supports one-way input messages that have no response, request/response, and one-way sends with or without a response. The last two patterns enable a service to specify other services that it needs.  

2.2.5 UDDI

UDDI provides a mechanism to register and categorize Web Services that you offer and to locate Web Services that you would like to consume. UDDI is itself a Web Service. Users communicate with UDDI using SOAP messages. UDDI acts as a mechanism for holding Service descriptions of available services. UDDI is often thought of as a directory mechanism, but it also defines a data structure standard for representing service description information in XML (Kreger) [19]. The data structures holding information about available services provide answers to the client queries like, what business is hosting this service and what kind of business is it? What products are associated with this service? With what categories in various companies and product taxonomies, is this business or its Web service associated? Are there other aspects of the service (such as Quality of Service) that can influence whether a requester would choose to invoke the service? What keywords can be provided so that it is easier to find this service? (Kreger,) [19].

2.3 Security in Web Services

Requirements for Achieving Security

The basic security concerns in Web Services as per Rosenberg and Remy [28], Chopra [5] and Westbridge technologies [37] can be summarized as follows:

- **Confidentiality** is the property that information is not made available or disclosed to unauthorized individuals, entities, or processes, and guarantees that the contents of the message are not disclosed to unauthorized individuals.

---

10 The details provided above are conceptual and are intended for understanding the concepts of WSDL. It is nowhere necessitated to present such an in-depth level information when referring to them normally.
• **Data integrity** is the property that data has not been undetectably altered or destroyed in an unauthorized manner or by unauthorized users thereby ensuring that the message was not modified accidentally or deliberately in transit.

• **Identity:** It involves two things
  1. **Authentication** involves comparing provided assertions of information to something already stored about the individual. It is divided into three types,
     a. **Something you know:** Pin, Password, Pass Phrase, Shared Secret.
     b. **Something you have:** Key, Card, Token
     c. **Something you are:** Biometrics such as fingerprint etc.\(^{12}\)
  2. **Authorization** is the granting of authority, which includes the granting of access based on access rights and guarantees that the sender is authorized to send a message. When these two are achieved they contribute an identity to an individual.

• **Proof of origin** is the evidence for identifying the originator of a message or data. It asserts that the message was transmitted by a properly identified sender and is not a replay of a previously transmitted message. This requirement implies data integrity.

• **Non-repudiation** is the constraint that forbids someone from denying that they received confidential information. (Rosenberg and Remy) [28].

What happens if these concerns are not seriously considered?

What if a Web Service lacks these issues of security? O’Neill [24] and Thomas [32] present B2B and B2C transactions as examples to bring out the point of e-fraud where the bank credit details are transferred via the internet. If confidentiality is not provided then unauthorized user can get access to other users’ details. If Integrity of the web service user’s data is not taken care of, then unauthorized person can claim access to the actual user’s data and demand claim of the service provided. All these acts lead to e-fraud. Fraud at this level though severe can be acceptable but with the scenarios where B2B transactions involving heavy cash inflows and outflows may cause an unacceptable damage (Shinder) [29].

### 2.4 Attempts made to combat the concerns in Web Services

#### 2.4.1 Confidentiality

The first issue regarding security is **Confidentiality**. Rosenberg and Remy \(^{12}\) [28], provides two alternatives to this problem. They can be explained as follows:

- Shared Key Technologies
- Public Key technologies.

##### 2.4.1.1 Shared Key Technologies

This technology is sometimes referred to as symmetric encryption or secret key encryption because the same key is used to both encrypt and decrypt the message and this key must be kept secret from all non-participating parties to keep the encrypted message secret.

\(^{12}\) Rosenberg and Remy (2004) [28]: refer to first chapter in this book for more information these concepts.

\(^{13}\) Refer to chapter 3 pg 72 “Shared Key Technologies” paragraph.
Figure 4: Shared Key Concept. (Rosenberg and Remy) [28]

Figure 4 shows the working of the shared key concept. The primary issue of concern under this scheme is regarding the sharing of the common key. Two suggested alternatives are either through phone or through mail. None of these adhere to the goals of Web Services. Shared key encryption uses cryptographic algorithms known as block ciphers. This means that the algorithms work by taking the starting plaintext message and first breaking it into fixed-size blocks before encrypting each block. Two strong encryption algorithms used throughout the software industry are part of the XML Encryption standard: the long-standing 3DES and the relatively new, but highly reviewed and Analyzed Encryption Standard (AES) and Triple Data Encryption Standard (3DES) algorithms are used for encryption.

2.4.1.2 Public Key Encryption
Public key encryption is also referred to as asymmetric encryption because two different mathematically related keys are used to encrypt and decrypt the messages. This provides the facility for secure key exchange. Figure 5 shows the concept behind public key encryption. Public key encryption is based on the mathematics of factoring large numbers into their prime factors. This problem is thought to be computationally intractable if the numbers are large enough.

The authors (Rosenberg and Remy) [28] feel that there is no proper mechanism to communicate the keys. Kerberos distributing shared keys could be applicable but it is restricted only to closed environments where all principals requiring keys share direct access to trusted Key Distribution Centers (KDCs) and all principals share a key with that KDC.
Public key encryption can be applied only to small messages and this helps in achieving the goal of distributing shared keys.  

**Table 2: Comparison of two technologies used for achieving Confidentiality**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td>Shared</td>
<td>Public</td>
<td>Faster</td>
<td>logical bit operations</td>
<td>Same</td>
<td>Encrypt/Decrypt Message</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Slower (RSA 1000 times slower than DSE)</td>
<td>modular exponentiation</td>
<td>Different</td>
<td>Encrypt/decrypt Shared key</td>
</tr>
<tr>
<td>Operation On Large Message</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speed</td>
<td>Faster</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation on Large Message</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operational Logic</td>
<td>Same</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Keys Used for Encryption and Decryption</td>
<td>Same</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Padding of Spaces</td>
<td>Not Needed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common Usage</td>
<td>Encrypt/Decrypt Message</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 2.4.2 Integrity

Many authors suggest the use of Digital Signatures to achieve the concept **Integrity**. Digital signature involves utilizing a one-way mathematical function called **hashing** followed by **public key encryption**. The basic idea is to use a hash function to create a **message digest** of fixed and short length and then to encrypt this short message digest. A **message digest** is a short representation (usually 20 bytes) of the full message. You need to do this because, as you have just seen, public key encryption is slow and is limited in the size of message it can encrypt. A **hash is a one-way mathematical function that creates a unique fixed-size message digest from an arbitrary-size text message**. One-way means that you can never take the hash value and re-create the original message. Hash functions are designed to be very fast and are good in avoiding collisions i.e., producing different keys for different messages. **Uniqueness** is critical to make sure an attacker can never just replace one message with another and have the message digest come out the same anyway; that would essentially ruin the goal of providing message integrity through digital signatures (Rosenberg and Remy) [28]. They support the belief that it even provides non-repudiation as they quote, “The purpose of digital signatures is to provide a mechanism for message integrity (no one has changed the message since it was created) and non-repudiation (you cannot refute that this message exchange occurred). XML Signature enables you to encode digital signatures into XML.”

---

14 To establish identity and support non-repudiation (for example, in digital signatures) where the public key is used for encryption, decryption takes substantially longer than encryption. In integrity applications (also in digital signatures) where the private key is used for encryption, it is the other way around. This imbalance would be a problem when applied to large messages but is not an issue when applied only to small messages such as the 200-bit key being exchanged to enable shared key encryption or the 20-byte message digests used in digital signatures, which will be discussed shortly.

All Points written in this context are taken from the literature of (Rosenberg and Remy, 2004) [28]. For further information refer to pgs 72-80 of [28].

15 Refer to page 80 of [28] for more information on this.
If the signature obtained after the process shown in the Figure 6 is valid then integrity is said to be achieved else the data sent is assumed to be tampered by a middleman.

2.4.3 Identity

Newcomer and Lomow [21] write “A security token is a credential that proves identity. When using Kerberos, X.509, Security Assertions Markup Language (SAML), or XrML, you've already proven your identity at least once”, but does not really mean that identity is provided, these are only mere assertions. These assertions give authorization which in turn improves trust and as the trust improves identity improves.

The basic feature is the use of secured username and password. It is difficult for one to crack these details and hence they are assumed to provide a proper identity. On top of it when encryption is used it makes further difficult for one to decrypt the details of these tokens. A subsidiary method which can be used is the use of X.509 certificate entrusted to the service requester by the service provider or any other third party. The next way would be through the use of SAML which works on three XML-based mechanisms:

- Assertions: An XML schema for assertions made on user’s behalf.
- Protocol: An XML schema and definition for a request/response protocol.
- Bindings: Rules on using assertions with standard transport and messaging frameworks. These rules are described as a set of bindings and protocols.

Another approach followed by Microsoft is the use of passports. Amongst these, passports and SAML have not attained global attention yet as there are no clear rules defined for these. Recently OASIS committee came up with a standard document for this technology and a lot rests on the results of imposing these standards. Kindly refer to the literature on these topics to gain more knowledge on these topics.
2.4.4 Non-Repudiation

According to Dournaei[16][8], an RSA signature provides non-repudiation and data integrity in two ways. The signer is the only person who can create the signature because only they have the private key. This means that a signer cannot deny that a message was signed with their private key; this is called non-repudiation and provides authenticity of the signer provided that there is an accurate binding between the signing key and the actual individual. Data integrity for the document is provided by the combination of the RSA algorithm and the SHA-1 hash function binding between the signing key and an actual individual.

This is to an extent does answers the problem of non-repudiation. This is not a comprehensive solution to the problem. It should be followed by the either of the two ways:

- Carry out the same procedure from client side to ensure that the client cannot deny the service (or)
- Assert a third party assertion to sender that the client received the service that he/she intended for.

2.5 Basic Mechanisms to Handle Message Security

There are some techniques like SSL, TSL and VPNS etc., but Yang[17][39] raises the same old questions that one has “Can I rely on Secure Sockets Layer (SSL), Virtual Private Networks (VPN), and dedicated lines to protect my traffic?”, the author along with O’Neill [24], Rosenberg and Remy[18][28] feel that none of these technologies fail to answer the case when the message is to be transferred to the client through multiple clients. It is with these questions in mind this section explains the position of SSL in current day’s Web Services environment.

Each one of the cited authors present the problem in their own way, and discussing it entirely would not be possible in the current work but an attempt is made to bring out the issues motivating the need for suggesting a new Security standards. The reasons presented in subsection 2.5.2 are some amongst those that mooted the need for a new message security revolution which led the OASIS (Standardization committee) to come up with a draft for message security in Web Services in 2004.

2.5.1 The role of SSL

Many people expressed many questions and doubts regarding the pole of SSL in current Web Services paradigm and there is quite a need to address this issue before concluding this session. The need is that authors like Rosenberg and Remy [28] believe that Encryption, Signatures and SSL and various other technologies are quintessential to cover the circle of security. On the contrary there are even sceptics like O’Neill [24], Yang[19][39] who believes that though it is necessary but not sufficient to meet the current day security needs.20

Rosenberg and Remy [28] present a good description of SSL and express their approval for its effective security measures for point-to-point or peer to peer transport. Yet they express concerns regarding its performance in Web Services and feel that it is new scenario all

---

[16] For more information on the working of RSA etc. kindly refer to Pg 34 & (24-36).
[17] Refer to page number 20 of the volume, right hand side first paragraph.
[18] Refer to pg 101 of the text book.
[19] Refer to page number 20 of the volume, right hand side first paragraph.
[20] A reference to this point (last sentence in above paragraph) is also made in the motivation Section of the article as it the primary Aim of the work is to check how far is the OASIS draft on WS Security (March 2004) is supported by the currently available Web Services tools.
together. They enlist four options, out of which O’Neill [24], raise doubts amongst prevalent and generally referred first two. The four options suggested are presented below followed by the concerns raised by O’Neill [24] and Yang[21] [39]. Four options are available when you are using SSL Transport Layer Security over HTTP (which you will see as HTTPS):

- **SSL/TLS (one-way):** SSL that one uses online when entering credit card details on a website. Using one-way SSL, you obtain two benefits:
  - Your client (browser) verifies the identity of the server.
  - You have an encrypted session between your client and the server.

- **Basic authentication:** With basic authentication, the client sends a username and a password for authentication. These credentials are sent in the clear format, so it is common practice to combine basic authentication with one-way SSL.

- **Digest authentication:** Digest authentication addresses the issue of the password being in the clear by using hashing technology (for example, MD5 or SHA1).

- **Client certificates (two-way or mutually authenticated SSL):** This is one-way SSL, as discussed previously, with the addition that the client must also provide an X.509 certificate to authenticate itself with the server. The protocol involves challenge and response between the server and the client in which information is digitally signed to prove the possession of the private key. This, in turn, is to prove that the identity based on the public key contained in the certificate can be trusted. This option is powerful, but it adds a great deal of complexity, especially for web applications with large numbers of consumer clients, because each client needs to be issued a certificate to gain access.

### 2.5.2 Concerns Raised On SSL

Yang[22] [39] asks “Can I rely on Secure Sockets Layer (SSL), Virtual Private Networks (VPN), and dedicated lines to protect my traffic?” the author discards SSL as an alternative for Web Services, as in Web Services an application can play a role depending on the context and business issues and feels that each application on its way either to access or render service opens the gateway to a potential hacker who is always lurking in the corner causing a threat knock.

O’Neill [24] presents an essay on the SSL and its liabilities. The author highlights few interesting facts, which are worth referring here:

- The very basic issue which dampens the significance of SSL is its strong binding to the routing and transfer of message content. The SSL or TSL no matter how good at providing security are impotent in Web Services scenario as the basic goal of Web Services is to provide flexible, robust and reliable architecture to share the data.

- Another fundamental requirement of a web service is that authenticate once and use till finish, i.e., once the server lets the user access the services until the author logs out of the session or there is loss of contact the user can use the privileges offered to him from the service provider. This is not acceptable with SSL which imposes user information every time a transaction is done.

- Web Services can communicate with each other without having to agree on common software packages, carry out extensive integration, or sign up for a third-party service provider which is not possible with basic SSL.

---

21 Refer to page number 20 of the volume, right hand side first paragraph.
22 Refer to page number 20 of the volume, right hand side first paragraph.
• With SSL authentication, the recipient only knows which entity is sending the SOAP request. In the real-world example shown in Figure 7, an end user authenticates itself to a website from its browser and fills out a web form to request information. This information is fetched, on the end user’s behalf, using a SOAP message to a Web service”.

![Figure 7 Model Of Web Services Architecture. O’Neill [24]](image)

• **Confidentiality** must also be applied to the SOAP data, which is in transit between the website and the web service. Session-layer security can be used to satisfy these requirements since SSL can encrypt both communication channels but it isn’t suitable to authenticate and authorize the SOAP request because the web service will only have visibility to one side of the communication (the website). SSL authentication provides no hint of the end user’s identity because the Web service won’t know on whose behalf the SOAP request is being generated. There is a chance for intruders to introduce malicious content into the service and disrupt the activity. One can refer to Cole et al. [6], King [18], Burns [3] for the type of effect these have and covering this aspect is not in the context of target work as these begin from Back door attacks, Spoofing, Man-in-middle, Replay, TCP/Hijacking, Fragmentation, Weak-Keys, Port scanning, Dampster Diving, Birthday attacks, Eavesdropping, War driving, etc.

• There are many more points which have really motivated the standardization of Web Services Security and presenting a thorough detail is an extensive work in itself. Yet the prime motive behind this reasoning of SSL should be understood to the user. That is mere usage of SSL or any other mechanism like VPNs or for that matter any other tool or technology would not suffice to the need of the hour and a combinatorial effort of some of the best aspects of these existing technologies with some newly founded ones is very vital for achieving the standards of Web Services security set.

### 2.6 Summary

This section is an attempt to present a brief description of what is being done or rather attempted to do so far to tackle Web Services security issues. To provide a comprehensive listing of all the works in the literature is beyond scope and vicinity of the work taken. One

---

23 If **two-way or mutually authenticated SSL** is used then this is not a problem but when SSL is referred in general it refers to only one way basic authentication SSL.

24 Interested readers are requested to kindly refer to chapter 17 of the text book Network Security Bible [6] for further information the topic.
must remember that this is still an active area of research and there is a cornucopia of material or literature on every subject described above.

- **XML Signatures** are needed to provide an overall **Integrity and Non-Repudiation**
- **Encryption and Decryption** are needed to provide **Confidentiality**
- **Assertions** are needed for **Authentication**
- **Identity** is provided by the third party certificates like **Kerberos, X.509 etc.**

Although these issues are handled in a pretty decent ways there does not exist a commendable standard for going through these aspects. This is what the coming sections try to point out. Apart from this the reader should have understood by now that the existing techniques like XML signatures, encryption though are necessary for the Web Services security, but are not sufficient to ensure required level of protection for messages. Thus, **WSS-2004** is a draft that tries to put forth some standards and the target of this work is to verify the support for that document from web service tools available. The coming sections will introduce these issues shortly to the reader.
3 What is WSS and what is its Contribution?

3.1 Potential Reasons for releasing WSS-2004

Readers are expected to understand the liabilities, pitfalls and susceptibilities in the current Web Services model by now. Continuing the journey from here, Web Services being End-to-End and not point-to-point.

- Are designed to work across any type of transport protocol, right from HTTP to FTP.
- Are complex and cumbersome, it does not involve a security mechanism that can be perceived in a straightforward way.
- The complexity allows each one to follow their own means to achieve the required target bringing back the grid from where it started from (i.e. proprietary issues, etc.)
- Finally and most crucially transfer message problems. These are explained in the coming subsection.

Chronology of Security

Figure 8 Protocols, Functions, threats and Mechanisms to Secure at various layers of OSI-ISO Model
The security issues that the current work aims at are the ones described in WSS-2004. These are issues related to the **message security** and hence only the issues pertaining to the vicinity of this area are discussed here and other issues like those pertaining to physical, data link layer, etc are ignored because discussing all the issues here is beyond the scope of the work, but even then it is presented here so that comprehensive listing of all the layers would be contributed to the readers interested. Let’s begin with a recollection of **International Standards Organization’s Open System’s Interconnect Model** (ISO OSI). Figure 8 shows the seven layers with each layer’s significance summarized as role, threat and mechanism to secure. For detailed explanation of the content refer to the numbered references in the [] provided.

In many ways the figure is self explanatory and text written in current context is primarily to aid understanding the shown concepts with some insight. Cole et al. [6] alone cites nearly more than 22 known types of attacks which disturb the Web Services. These begin with **viruses, denial-of-service (DoS) attacks, Back door, Spoofing, Man-in-middle, Replay, TCP/Hijacking, Fragmentation, Weak-Keys, Port scanning, Dampster Diving, Birthday attacks, Eavesdropping, War driving**2, etc and describing each of these is not possible in our article but short introduction to some of the crucial are defined and explained below. The threats which Web Services are concerned about fall in the Network, Transport, Session, Application and Presentation layers and they are:

1. **Disclosure of configuration data and message replay** (King [18]) – Message replay is a type of attack in which the attacker captures a legitimate message and later replays that message in order to gain unauthorized access to resources. Depending on how this attack is performed, it could be categorized under the “unauthorized access” threat described above. Disclosure of configuration data, in turn, could possibly be categorized under the “reconnaissance” threat described below.
2. **Network eavesdropping** (King [18]) – Interception of messages sent between the intended parties is always a threat when public infrastructures are used. Traditionally VPN or SSL have been used to protect data in transit. However, these techniques are not always adequate in order to protect a Web Service.
3. **Denial of Service** (Burns [3]) “**Domino effect**” **Denial of Service attack** (De Jesus [7]) – The complexity that several services coupled in a network may exhibit, as described in the prior section, may also lead to unwanted effects in the case of a “message bombing” aimed at one service in order to perform a DoS attack (De Jesus [7]). The “message bombing” may have ripple effects that may lead to unforeseen DoS-effects on other, dependent, services. DoS attacks can be very destructive in terms of limiting legitimate access to resources and they can be difficult to completely guard against.
4. **Reconnaissance** (Burns [3]) – Every expert hacker study their target carefully before launching an attack. The attractive feature of Web Services that allows a customer to search for an interesting service to use is equally a means for a hacker to use in order to gain intelligence about the potential victim. In addition to traditional sources of information like WHOIS databases and DNS servers, UDDI presents an excellent information source for hackers to use.

---

2 Interested readers are requested to kindly refer to chapter 17 of the text book Network Security Bible [6] for further information the topic.
5. **Unintended software interactions** (Burns [3]) – The complexity of Web Services mean that it will take a while before a sufficiently large body of knowledge exists that can be used for defining industry best practices with regards to security. This is similar to the problem mentioned below.

6. **Back door** (Cole et al. [6]) – A back door attack takes place using dial-up modems or asynchronous external connections. The strategy is to gain access to a network through bypassing of control mechanisms by getting in through a back door such as a modem.

7. **IP Spoofing/Masquerading** (Cole et al. [6]) – IP spoofing is used by an intruder to convince a system that it is communicating with a known, trusted entity to provide the intruder with access to the system. IP spoofing involves an alteration of a packet at the TCP level, and is used to attack Internet-connected systems that provide various TCP/IP services. In this, the attacker sends a packet with an IP source address of a known, trusted host instead of its own IP source address to a target host. The target host may accept the packet and act upon it.

8. **Man-in-the-middle** (Cole et al. [6]) – A man-in-the-middle attack involves an attacker, A, substituting his or her public key for that of another person, P. Then, anyone wanting to send an encrypted message to P using P’s public key is unknowingly using A’s public key. Therefore, A can read the message intended for P. A can then send the message on to P, encrypted in P’s real public key, and P will never be the wiser. Obviously, A could modify the message before resending it to P.

9. **TCP Hijacking** (Cole et al. [6]) – an attacker hijacks a session between a trusted client and network server. The attacking computer substitutes its IP address for that of the trusted client and the server continues the dialog believing it is communicating with the trusted client. Simply stated, the steps in this attack are as follows: 1. A trusted client connects to a network server. 2. The attack computer gains control of the trusted client. 3. The attack computer disconnects the trusted client from the network server. 4. The attack computer replaces the trusted client’s IP address with its own IP address and spoofs the client’s sequence numbers. 5. The attack computer continues dialog with the network server (and the network server believes it is still communicating with trusted client).

The attacks defined above are named differently in different literatures with slight or unalterable changes and it’s up to the reader to follow any of the convention. There are many mechanisms cited in the literature to overcome these potential threats. It can be seen from the Figure 8, that threats numbered 1,3,5,7,8 and 9 might be contained to some extent using mechanisms like AU’s, ESPs (Encrypted Authentication Headers & Encrypted Security Payload) and some others are like threat 6 and by-passing firewalls etc can be avoided by using port commands which restrict port 443,445 etc. The list of mechanisms and threats as written earlier is very vast and cannot be covered here but enthusiastic readers can refer to cited references for further detail.

To summarize this Section, the reader is introduced to the message transfer problems and threats to get a briefing of the steps taken by the network architects to provide Message Security till today, their performance to meet challenges and to achieve the required security. These suffer from the conventional problems (already described in other sections) like proprietary rights, lack of standardization, very finely refined nature of work etc. This ultimately prompted the IETF (Internet Engineering Task Force) and other web technicians to come up with a unique and once for all solution to the problems of message security in Web
Services at all levels of the architecture which could comprehensively defend any unintended attacks. People with great vision foresaw these issues and thought of standardizing the security architecture and hence WSS-2004 was released to answer the issues discussed in this section. (Rosenberg and Remy) [28]

### 3.2 Web Services Message Specification standard-2004

Before discussing anything the reader should kindly note that from here on WSS-2004 refers to this specification. The summary presented just above reasons for WSS-2004’s entry into the Web Services scenario. We have seen that technologies like SSL, TSL, Firewalls, PKI encryption Signatures, etc each one suffered at the hands of the intruders. The reasons sighted in literature can be anything but the ultimate aim of the Web Services Community and e-business believers is threat free, secure and safe environment for business.

The IBM and Microsoft [13] together published an architecture which covered a road map to secure Web Services of the future which according to Rosenberg and Remy [28] vision would be something like one shown in the Figure 9 showing the state of position then.

![Figure 9 Roadmap of Web Services Security](image)

Web Services Message Specification standard-2004 is the second layer of this paradigm and is expected to lay a strong foundation to remaining blocks. This section aims to introduce the reader to these issues before analysis phase to provide a firm understanding of the underlying thoughts and design aspects behind them.

As discussed in the last of the previous section people with great vision foresaw the upcoming problems and decided to standardize the paradigm so that the problem of web security is dealt with once and for all. Thus a roadmap as shown in the Figure 9 was in the back of the minds of the technical architects to curb any discrepancies that may arise in future due to flaws in the paradigm. The figure consists of various blocks that can be used as bricks to construct strong secure Web Services wall. One should understand that by further classifying in to blocks the WS-Security model introduces into Web Services what may be called as **independence between these blocks** as one can concentrate the details specific to a specific block rather considering the entire WS-Security model. Rosenberg and Remy [28] define issues as

- **WS-Policy**— defines how to express capabilities and constraints of security policy.
- **WS-Trust**— describes the model for establishing both direct and brokered trust relationships, including intermediaries.
- **WS-Privacy**— Enables users to state privacy preferences and Web Services to state and implement privacy practices.
• WS-Secure Conversation— describes how to manage and authenticate message exchanges between parties, including exchanging security contexts and establishing and deriving session keys.

• WS-Federation— describes how to manage and broker the trust relationships in a heterogeneous federated environment, including support for federated identities.

• WS-Authorization— defines how Web Services manage authorization data and policies.

One should note that the bulleted points above are yet to be standardized and we are at stage 2 of the road map in the Figure 10. When these bulleted points are standardized hopefully the web community can expect a strong security shield for Web Services model.

3.2.1 WSS: Old Actors in New Key-Roles

This section to find out how WSS intend to achieve the message security goals set for Web Services. The WSS-2004 came up with new innovations in the working of Web Services and implementing security. The primary fields which demand more insight from the reader are the changes brought about in

• Security Tokens
• Encryption
• Signatures

3.2.1.1 Security Tokens

Security tokens are security artifacts included in the message that are typically used for authentication or authorization purposes Rosenberg and Remy [28]. There are three types of commonly used security tokens namely,

• Username password tokens - It is used for username and password. These tokens are used to account for the cases where plain text usernames and passwords are used.

• Binary Security Tokens – They are in binary format and hence they are called binary security tokens. Two formats of credentials namely X.509 v3 and Kerberos use these. Base-64 encoding is used commonly to encode the credentials.

  → Binary Security tokens used with X.509 certificate
  Having the certificate does not authenticate the identity of SOAP sender. Digital signatures are the way you prove that you have the private key that was used to digitally sign a message. If the message's receiver is able to successfully verify the XML Signature with the public key in the X.509 certificate, and the receiver trusts the certificate authority that verified the identity associated with the private key that signed the X.509 certificate, an X.509 certificate becomes a strong authentication mechanism.

  → Binary Security tokens can be used with Kerberos Tickets
  It has two parts key granting ticket giving assertion of identity and a service ticket. These two tickets in combination contribute identity.

• SAML tokens - It is used for assertions. The relying part must be convinced either through sender-vouchers or sender’s key as in key the first type as their names suggest are concerned with usernames and passwords.

There are many more and one can refer to literature for more information on this.
3.2.1.2 Encryption
In simple XML encryption, one would replace the plain text with the encrypted text. Now with WS-security encryption a new element called a ReferenceList is added in the security header. It will point to the parts of the message that have been encrypted. In WSS-2004 the overall working scenario changes as follows, First, the message itself, or perhaps multiple individual parts of the message, are encrypted using a shared key. The shared key is then encrypted using the recipient's public key. The recipient possessing the private key can decrypt the shared key and then proceed to decrypt the rest of the message.

The technique is often called key wrapping or digital enveloping because the shared key is wrapped by the recipient's public key. One thing the reader must however not forget is that main SOAP elements i.e. Envelope, Header, and Body are never encrypted as they form the anchors on which SOAP messaging relies upon.

3.2.1.3 XML Signatures
As discussed in WS-Security tokens, one of the functions provided by signatures is providing identity and second thing is the means to provide shared key. One of the useful by-products is the guarantee of integrity through signature as redressing any small part of the signed content renders entire metadata as malcontent. There are three common practices of using signatures,

- **Enveloping Signature**: An Enveloping Signature wraps the item that is being signed.
- **Enveloped Signature**: In an Enveloped Signature, the reference points to a parent XML element, and
- **Detached Signature**: Detached Signature points to an XML element or binary file outside the Signature element's hierarchy.

Among the three types (Enveloped, Enveloping, and Detached) detached type of the signatures is well suited for the Web Services. Its worth to know some values such as,

- SignatureMethod is a pointer to the signature algorithm.
- The Reference elements are the pointers to what is being signed.
- DigestMethod contains the one-way hash algorithm.
- SignedInfo is the XML block representing the information that will be signed.
- SignatureValue element is a digital signature of the SignedInfo block.
- KeyInfo block is one that either contains the key to use for verifying the signature or has information necessary to look up such a key.

3.2.2 The Elements of WSS- 2004
The specification came up with roughly eight elements with few mandatory implications and few optional implications on these elements. These are shown in the Figure 10 below. WSS-2004 draft leverages the core issues of the security on these eight elements. Rosenberg and Remy [28] and Dourmaee [8] probe in detail about these issues. To understand these in a better way lets begin with Namespaces.
3.2.2.1 Namespaces

It is a mechanism that provides uniqueness to the names used in a document. It is very much analogous to the packages constructed in C++. The uniqueness is achieved through the usage of URI. As per Rosenberg and Remy [28], URI’s can be summed up as something like

\[ \text{URN} + \text{URI} = \text{URI} \]

This is done because URN names a resource uniquely but it cannot be dereferenced and referenced and URL though can be referenced and dereferenced does not ensure that they are registered, and for uniqueness it is mandatory to register a part or whole of the URL with some authority like IANA (Refer to literature for more information on URL and URN). When this is done the product obtained ensures uniqueness. Prefixes are added to address these in an easier way in the XML document. The need for using this can be explained by the fact at a time a reference can used in four files like

- The instance document carrying the data,
- The SOAP envelope defining the message format,
- The WSDL instance document describing the interface, and
- The WSDL schema validating the interface definition. This is the minimum number of documents involved with a Web Services conversation; others can be added depending on the service.

To present a better understanding of these things look at the examples cited below.

<table>
<thead>
<tr>
<th>Prefix</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>ds</td>
<td><a href="http://www.w3.org/2000/09/XMLdsig#">http://www.w3.org/2000/09/XMLdsig#</a></td>
</tr>
<tr>
<td>S11</td>
<td><a href="http://schemas.xmlsoap.org/soap/envelope/">http://schemas.xmlsoap.org/soap/envelope/</a></td>
</tr>
<tr>
<td>S12</td>
<td><a href="http://www.w3.org/2003/05/soap-envelope">http://www.w3.org/2003/05/soap-envelope</a></td>
</tr>
<tr>
<td>Wsse</td>
<td><a href="http://docs.oasis-open.org/wss/200401/oasis-200401-wsswssecurity-secext-1.0.xsd">http://docs.oasis-open.org/wss/200401/oasis-200401-wsswssecurity-secext-1.0.xsd</a></td>
</tr>
<tr>
<td>wsu</td>
<td><a href="http://docs.oasis-open.org/wss/200401/oasis-200401-wsswssecurity-utility-1.0.xsd">http://docs.oasis-open.org/wss/200401/oasis-200401-wsswssecurity-utility-1.0.xsd</a></td>
</tr>
</tbody>
</table>
3.2.2.2 Id Reference
Before venturing into the details of this element, let's have some notes on the background of XML and XML schema. The scope of any reference previously, rested in and was in fact mostly restricted to the underlying XML schema of the web service tool or provider. This may some times cause problems if there are structural elements (XML nodes or sub nodes) with same identity as it results in garbled reference. To alleviate this problem WSS-2004 came up with an innovative scheme of ID references which primarily separated IDs into what is called as Local IDs and Global IDs.

How can this help intermediaries solve garbled references?
The specification answers this problem as, “To simplify the processing for intermediaries and recipients, a common attribute is defined for identifying an element. This attribute utilizes the XML Schema ID type and specifies a common attribute for indicating this information for elements.
The syntax for this attribute is as follows:
<anyElement wsu:Id="...">...
</anyElement>”
The above citation suggests that, every XML schema often has some sort of a tag for providing the description of the type called “type” (also referred as attribute type) of the field and it is suggested in those severe conditions one can use the underlying XML schema to caulk any threats of garbled references.

3.2.2.3 Security Header
The <wsse:Security> header block is a mechanism for attaching security-related information targeted at a specific recipient in the form of a SOAP actor/role. This may be either the ultimate recipient of the message or an intermediary. Consequently, elements of this type may be present multiple times in a SOAP message. There can be multiple Security headers in a SOAP Message if one really requires them (WSS-2004). The beauty of this component is that it facilitates flexibility on part of SOAP header.

3.2.2.4 Security Tokens
The WSS-2004 specification provides an <wsse:BinarySecurityToken> element for security tokens. The <wsse:BinarySecurityToken> element defines two attributes that are used to interpret it. The ValueType attribute indicates what the security token is like Kerberos ticket and the EncodingType tells how the security token is encoded, like base64 encoding. The document defines mechanisms for identifying and referencing security tokens using the wsu:Id attribute and the <wsse:SecurityTokenReference> element.

3.2.2.5 Token References
This element is addressed in the document to generalize the way in which the tokens are referenced, as all tokens are not referenced in the same way.
The references can be classified into following types.
- **Direct References** –
  1. One can use URLs fragments to refer to the included content within some content, and
  2. Full URIs to refer to the external tokens
• **Key Identifiers** – This allows tokens to be referenced using an opaque value that represents the token (defined by token type/profile) i.e., it is like an alias to refer to the actual content.
• **Key Names** – This allows tokens to be referenced using a string that matches an identity assertion within the security token. This is a subset match and may result in multiple security tokens that match the specified name.
• **Embedded References** - This allows tokens to be embedded within the element itself.

### 3.2.2.6 Signatures

This is the most crucial part of the specification as it provides answers to many of the underlying security threats like origin of message and integrity. In short “XML Signature protects parts of an XML document from unauthorized modifications using a cryptographic technique called digital signature. The signature algorithm can be either a symmetric cryptosystem such as Triple DES and AES (Advanced Encryption Standard), or an asymmetric (public key) cryptosystem such as RSA and DSA” (Rosenberg and Remy) [28]. These procedures are already described and hence are not covered in detail in this section.

They are used to verify message origin and integrity. This is a way through which message producers demonstrate their knowledge of the key, typically from a third party, used to confirm the claims in a security token and thus to bind their identity (and any other claims occurring in the security token) to the messages they create (WSS-2004). The interesting part of the Signature is the role it plays in conjunction with the Security tokens because, they as a combination provide authentication and this is made crystal clear in the WSS-2004 document from the lines, “An authority can vouch for or endorse the claims in a security token by using its key to sign or encrypt (it is recommended to use a keyed encryption) the security token thereby enabling the authentication of the claims in the token”. Some examples of signed security tokens:

1. X.509 Certificates
2. Kerberos Tickets

Addressing integrity and confidentiality concerns from the specification side, it states that the specification “provides a means to protect a message by encrypting and/or digitally signing a body, a header, or any combination of them (or parts of them). Message integrity is provided by XML Signature [XMLSIG] in conjunction with security tokens to ensure that modifications to messages are detected. The integrity mechanisms are designed to support multiple signatures, potentially by multiple SOAP actors/roles, and to be extensible to support additional signature formats. Message confidentiality leverages XML Encryption [XMLENC] in conjunction with security tokens to keep portions of a SOAP message confidential. The encryption mechanisms are designed to support additional encryption processes and operations by multiple SOAP actors/roles. This document defines syntax and semantics of signatures within a <wsse:Security> element.”

The working model is same, a producer sends the base64 encoded security token which contains Symmetric key and it is expected that they are properly authenticated between themselves through the use of PKI, etc then the recipient on reception of the binary encrypted day uses his/her shared key (or symmetric key) to get the plain text. The author then verifies it with the actual content sent. When they both tally he/she goes ahead with further procedure. Other details regarding encryption in the citation above is covered in the Section 3.2.2.7 below and also in the main introduction of the work which could be referred for further understanding. The thing which one should understand from the citation above is that the
document only provides syntax and semantics (guidelines) for secure Web Services and nothing more than that.

### 3.2.2.7 Encryption

After the signature this is the next most crucial element on which the security of a web service rests. The specification includes this segment in the document in order to add **flexibility** dimension to existing encryption model using the three elements of encryption namely `<xenc:ReferenceList>`, `<xenc:EncryptedData>` and `<xenc:EncryptedKey>`. The `<xenc:ReferenceList>` is used to describe the content that is or would be encrypted. The `<xenc:EncryptedData>` contains the digest of the content encrypted and `<xenc:EncryptedKey>` is used to specify the key details used for encryption.

### 3.2.2.8 Security Timestamps

This element was included into the document in response to the freshness aspect of the message. The `<wsu:Timestamp>`, element provides a mechanism for expressing the creation and expiration times of the security semantics in a message. These are very handy in scenarios where replay attacks and denial of services attacks occur. An approximate estimate of period is considered for valid information and any packets arriving or data that is received after the passage of that period from the time stamp in the SOAP content is discarded.
4 Achieving the Objectives

4.1 Objective 1: Understand WS-Security 2004

The first objective of understanding is accomplished through reading and analyzing the document. The fulfillment of this objective is evident from the Section 0 where a descriptive explanation of the document is provided.

A brief explanation of rating a particular aspect as per personal conception is presented. Some of the ratings could not have conclusive evidences due to some unexpected liabilities. The contexts and liabilities encountered are presented in the description. The reader has to reference various listings during this process because including the listings exceeds the text limitation imposed on the document.

4.2 Objective 2: Selection of Web Services tools

Three Web Services tools were selected namely BEA SP 8.12, Visual Studio .Net 2005 Beta 2, and Apache Axis Server 1.2.1 is used. The reasons for selecting these tools:

- BEA Weblogic 8.12 was chosen for testing because the hardware configuration that the system had was not enough for the other advanced BEA versions. For the hardware details refer to Appendix D (System Information). Bea provides free copy of the software which is also a reason for BEA being selected. Finally and most importantly BEA is one of those Web Services tools that have widespread reputation. So evaluating it would contribute more to Web Services society as more part of the web community is expected to be benefited.

- Visual Studio 2005 beta 2 was chosen although it was a beta version because it was apart of the agenda that one Microsoft programming platform and one non- Microsoft programming platform were to be used for generating SOAP content. Hence Java and C# were chosen for code development and to carry out the evaluation. Secondly, Microsoft is one amongst the forerunners in establishing Web Services and security standards thus more knowledge or details are expected to be revealed during its evaluation which may be useful in future work.

- Ultimately Apache Axis Server 1.2.1 was used because it is very new. Secondly, it is open software hence details regarding architecture and technology used were readily available. It even has a new package called WSSJ whose feature allures one to gain insight regarding it.

- The tools from vendors such as IBM Websphere, Systinet, etc are not used due to lack of specified hardware requirements and commercial reasons.

4.3 Objective 3: Determination of rating procedure

The rating procedure can be summarized into four points. The first point is formulation of cases and codes from the WSS-2004. These codes serve as token bearers in the results table. So the constraints specified in WSS-2004 are classified into codes and are placed in first column of the results table (Table 3). How these codes are formed is shown in Section 5.1. The sample tutorials taken from references provided are implemented on the corresponding tools. The SOAP message content is then taken from the log files stored in the logs directory of the tool (Every tool stores log files in a specific directory for information on these directories refer to the documentation available at the tool’s site). As per the observations of
the SOAP messages (shown in the Appendix B: Listings), a rating is given against the code in the results table. The reasoning for the rating is also given and reader can refer to these reasons in Appendix A.

4.4 **Objective 4: Analysis of Results**

The analyses of the results are presented in the Section 6.2. The reader is requested to refer to that section for more information.
5 Research Study
The research approach steps are listed as following. The discussion in this section follows these steps.

1. To understand the constraints and standards imposed by the OASIS Message Security
draft [23] (WS-Security 2004) and formulate codes (cases) from constraints that can be
used as token bearers to hold the rating during analysis phase
2. To reason why only certain tools were chosen for evaluation though there are many
vendors providing the Web Services tools.
3. To determine how should the rating be done.
4. To get the Web Services tool’s SOAP message content because message standard itself
   is SOAP Message based. If a constraint could not be verified through SOAP message
   content what other means could be used to verify.
5. To check for the demanded constraints in SOAP message and present the rated results.
6. To analyze the ratings given.

The first two objectives and these steps overlap. This is only coincidence and hence
these steps (i.e. why certain tools were selected and understanding of specification draft) are
not explained in this chapter as it is expected that the reader knows what is being done with
these steps from the explanation provided regarding first two objectives in Sections 4.1 and
4.2. The third step is explained in the Section 5.1.

5.1 To determine how should the rating be done
In this section all the points that have to be verified from the WSS-2004 are collected and are
classified into codes. Every mandatory issue is referenced by a code as shown in the
respective subsection. The results table, Table 3 holds these codes as the first column elements
and other columns are headings of the Web Services tool names.

The coding convention followed is very simple. There are four categories of mandatory
issues, Must, Must Not, Should and Should Not. The category (like must, etc) and issue
standardised in the WS-Security 2004 like Security Tokens, Encryption, etc are concatenated
through a – mark to constitute a code. The code contains all the points of the category (must,
must not ...) included under the standard (Security token, TimeStamp, etc). According to the
observations and intuitions, known or developed, a rating is placed in the respective tool’s
column as a measure of its support. The legends used in the table are explained what they
stand for in the context of the table right under the table itself. The details of the codes used in
the table are provided below.

5.1.1 Namespaces Mandatory Issues
‘The Namespaces listed below in the footnote are mandatory and have code Must-
NameSpaces in the Results table.’

26 These are the Namespaces mandated

http://www.docs.oasis-open.org/wss/200401/oasis-200401-wss-wssecurity-secext-1.0.xsd
http://www.docs.oasis-open.org/wss/200401/oasis-200401-wss-wssecurity-utility-1.0.xsd
5.1.2 ID Reference Mandatory Issues

Coming to the ID Reference mandatory issues, it accentuates that every one of the wsu:Ids used must be unique. It is covered under code Must Not-Ids in the results table. In reference to the point in WSS-2004, “Implementations MAY rely on XML Schema validation to provide rudimentary enforcement for intra-document uniqueness. However, applications SHOULD NOT rely on schema validation alone to enforce uniqueness”. It should be noted that this cannot be evaluated in the current work as the schema followed by a tool is a proprietary issue and tool users can very seldom have access to these details.

5.1.3 Security Header Mandatory Issues

The security Header mandatory issues include:

- Message security information targeted for different recipients MUST appear in different <wsse:Security> header blocks.
- When a <wsse:Security> header includes a mustUnderstand="true" attribute:
  1. The receiver MUST generate a SOAP fault if does not implement the WSS: SOAP Message Security specification corresponding to the Namespace. Implementation means ability to interpret the schema as well as follow the required processing rules specified in WSS: SOAP Message Security.
  2. The receiver must generate a fault if unable to interpret or process security tokens contained in the <wsse:Security> header block according to the corresponding WSS: SOAP Message Security token profiles.
- Two <wsse:Security> header blocks MUST NOT have the same value for S11:actor or S12:role. The <wsse:Security> header block without a specified S11:actor or S12:role MAY be processed by anyone, but MUST NOT be removed prior to the final destination or endpoint.
- Due to the mutability of some SOAP headers, producers SHOULD NOT use the Enveloped Signature Transform defined in XML Signature. Instead, messages SHOULD explicitly include the elements to be signed. Similarly, producers SHOULD NOT use the Enveloping Signature defined in XML Signature [XMLSIG].

It is covered under codes Must-Security Headers, Must Not-Security Headers and Should Not-Security Headers in the results table.

5.1.4 Security Tokens Mandatory Issues

Coming to the mandatory implications of these,

- The receiver must generate a fault if unable to interpret or process security tokens contained in the <wsse:Security> header block according to the corresponding WSS: SOAP Message Security token profiles.
- All compliant implementations MUST be able to process the <wsse:UsernameToken> element.

It is covered under code Must-Security Headers in the results table.

5.1.5 Token References Mandatory Issues

Most of the points suggested in this section are primarily under recommended severity level, but if the recommended levels are followed then they should follow them as per the standards described. That is either the tool providers can completely ignore them, but if they consider them then they have to adhere to the standards specified. Token attribute does not indicate the
ID of what is being referenced, that SHOULD be done using a fragment URI in a
<wsse:Reference> element within the <wsse:SecurityTokenReference> element. The
<wsse:KeyIdentifier> element SHALL be placed in the <wsse:SecurityTokenReference>
element to reference a token using an identifier. This element should be used for all key
identifiers (covered under Should-Token References).

5.1.6 Signatures Mandatory Issues
The mandatory issues in signatures are listed below.
• A message recipient SHOULD reject messages containing invalid signatures, messages
missing necessary claims or messages whose claims have unacceptable values. Such
messages are unauthorized (or malformed).
• To add a signature to a <wsse:Security> header block, a <ds:Signature> element
conforming to the XML Signature specification MUST be prepended to the existing
content of the <wsse:Security> header block, in order to indicate to the receiver the
correct order of operations.
• All the <ds:Reference> elements contained in the signature SHOULD refer to a resource
within the enclosing SOAP envelope as described in the XML Signature specification.
• If a producer wishes to sign a message before encryption, then following the ordering rules
laid out in Section 5, "Security Header", they SHOULD first prepend the signature element
to the <wsse:Security> header, and then prepend the encryption element, resulting in a
<wsse:Security> header that has the encryption element first, followed by the signature
element:
  <wsse:Security> header
  |encryption element|
  |signature element|
• Likewise, if a producer wishes to sign a message after encryption, s/he SHOULD first
prepend the encryption element to the <wsse:Security> header, and then prepend the
signature element. This will result in a <wsse:Security> header that has the signature
element first, followed by the encryption element:
  <wsse:Security> header
  |Signature element|
  |encryption element|
It is covered under code Must-Security Headers and Should-Security Headers in the results
table.

5.1.7 Encryption Mandatory Issues
The mandatory encryption issues are discussed below.
• The encryption step involves encrypting elements or element contents within a SOAP
envelope with a symmetric key, which is in turn to be encrypted by the recipient’s key
and embedded in the message, <xenc:EncryptedKey> MAY be used for carrying such
an encrypted key. This sub-element SHOULD have a manifest, that is, an
<xenc:ReferenceList> element, in order for the recipient to know the portions to be
decrypted with this key. An element or element content to be encrypted by this
encryption step MUST be replaced by a corresponding <xenc:EncryptedData>
according to XML Encryption. All the <xenc:EncryptedData> elements created by
this encryption step SHOULD be listed in the <xenc:ReferenceList> element inside
this sub-element, i.e., when digital signatures are used then its mandatory to use
\textit{<xenc:ReferenceList> should be used holding the information regarding the data}
\textit{encrypted and the encrypted data should be in the <xenc:EncryptedData>}
\begin{itemize}
  \item The message creator MUST NOT encrypt the \textit{<S11:Envelope>, <S12:Envelope>,}
  \textit{<S11:Header>, <S12:Header>, or <S11:Body>, <S12:Body> elements but MAY}
  \textit{encrypt child elements of either the <S11:Header>, <S12:Header> and WSS: SOAP}
\end{itemize}
It is covered under code Must-Encryption, Should-Encryption and Must Not- Encryption in
the results table.

\subsection{5.1.8 Security Time Stamps Mandatory Issues}

The time stamp mandatory issues are presented below.
\begin{itemize}
  \item Implementations MUST NOT generate time instants that specify leap seconds. If,
  however, other time specifier types are used, then the \texttt{ValueType} attribute (described
  below) MUST be specified to indicate the data type of the time format.
  \item Requesters and receivers SHOULD NOT rely on other applications supporting time
  resolution finer than milliseconds.
  \item All times MUST be in UTC format as specified by the XML Schema type \texttt{dateTime}. It
  should be noted that times support time precision as defined in the XML Schema
  specification.
\end{itemize}
It is covered under codes Must-Time Stamps, Should Not-Time Stamps and Must Not- Time
Stamps in the results table.

\subsection{5.2 To Get the SOAP Message Content}

When the server is up and running, the server periodically writes its status, errors generated,
SOAP content, and other detailed dynamic information in a file called log file. These log files
are stored in a separate directory for logs and its location depends on the installation of the
details of the Web Services tools. These files are text based files and when these files are
opened one could find the generated SOAP content. Apart from this in BEA and Microsoft
tools the browser used for running client side application can also be used to display the SOAP
message through a button provided on the web page. Refer to the documentation for tools at
the website provided in the Appendix A for more information.

\subsection{5.3 Check the demanded constraints and present the results}

Chapter 6 presents the results and analysis. Appendix A: Rating Reasons presents how the
rating is given to the codes in the table. The implementation details are as follows:

\textbf{Test Bench}

To begin with BEA, the tool’s website provides documentation about the tool according the
version. The version of the tool used is SP1.2. It is known from Rosenberg and Remy [28] that
there is some work already being done in BEA. A sample application was needed, to check the
support provided by BEA. It can be seen from the Figure 11 after installing the tool phase, to
implement a service, a tutorial or a sample is needed. For this purpose a web service sample is
selected from the documentation at the tool’s web site and implemented on the tool as per the
selected sample’s tutorial.
For the details regarding the sample refer to the Appendix C where references and details of the sample are provided. The snapshots of the monitor’s screen while implementation is provided in the Appendix C as this tool mostly works with GUI based environment. The web page in the browser provided by the tool displays the SOAP generated message when the tutorial of the sample is successfully implemented. Though the tutorial is for checking the severity of the security provided, in this work the SOAP message is checked from the standards perspective for constraints. This is concerned with the fourth step in the Figure 11, where SOAP message content is to be checked. Coming to the final step in the Figure 11 the results and the conclusions reached are, shown and explained in the 6 chapter. The same approach was followed with Microsoft Visual Studio tool. MSDN\textsuperscript{27} which stands for Microsoft Developer Network provides every detail of the Microsoft’s product and documentation on tutorials. This site along with the reference provided in the Appendix C are referred to get a Web Services sample required for evaluation of the specifications and implemented in the Microsoft Visual Studio editor. The browser used for executing the sample is Microsoft Internet Explorer 6.0. Note for BEA, the browser is generated by BEA tool itself but when BEA is checked in Linux system (as client) Firefox 1.0.7 is used. Analogous to BEA the results and conclusions reached are presented as per the reasoning provided in Appendix A. Note that the SOAP content would be available on clicking the service description button on the web page when the code is executed. Finally coming to the Apache Axis Server, the server needs Jakarta tomcat version higher than version 1.5, this is comparatively new tool and the framework it follows is called as WSS4J. The Sample is taken from the site referred in the Appendix C. This was done because this is a new tool and includes highly advanced programming constructs regarding XML programming. An initial attempt was made to construct a service as per Java Web Services tutorial 1.6 (JWSDP doc), but the Apache’s website recommended people lacking prowess regarding Java Beans architecture not to try their hands at coding and advised to follow the sample tutorial and code given in the code Appendix C. The reference from where the sample was taken is provided in the Appendix C. The log files are stored in the server’s log directory which holds the generated SOAP.

\textsuperscript{27} http://msdn.microsoft.com/
messages and error content. The SOAP message from the log file is taken and checked for the constraints. Later the conclusions drawn are briefly explained in the appendix A.

The System details are provided in the Appendix D called system information. The network used for the Web Services consisted of a LAN (Local Area Network) with single switch which included a server and two clients. The client machines hardware details are similar to the one on which the server was running. The protocol used to invoke the service is hyper text transport protocol (http). The speed of the LAN was 100 mbps. Note at one time only one server was run because of the small size of RAM (Random Access Memory) which is of 128 megabytes.

The evaluation or core content of the work, the checking and reasoning of the rating is explained in next Section 6.2. Kindly go through this section for details. The rated table is presented first and then the explanation to the rating is provided. There is no explicit reasoning for presenting the final work this way except that it allows one who is not interested in reasoning and is concerned only about rating could directly refer the ratings and skip going through the reasoning content, and one who is interested in reasoning content could refer to the correct code against the content he is interested to know. That is if one does not wish to know about security standards in Namespaces or signatures and wishes to know about only details regarding timestamps or encryption he could check the legend in the table below and refer to the reasoning provided.
6 Results, Evaluation and Analysis

The following Section 6.1 presents the results and evaluation in brief and the Section 6.2 presents the analysis our work.

6.1 Results and Evaluation

The Table 3 shows the ratings adjudged for various mandatory issue codes for three different tools: BEA SP1.2, Visual Studio 2005 Beta 2, and Apache Axis 1.2.1. The observations that led to these ratings are given in Appendix A. One can refer to these reasons for more details regarding ratings. The patterns in the results are explained in the section below. Majority of the specifications which are supported can be attributed to the XML schema being used by the tool. These schemas are proprietary and can be updated as per requirements. Other issues which needed details regarding proprietary issues are ignored due to lack of access to these details but the similar issues in open software like Apache Axis are analyzed and the details of reasoning are given in Appendix A.

Table 3: Analysis Result Table

<table>
<thead>
<tr>
<th>Final Analysis Of the Tools</th>
<th>Tool</th>
<th>BEA8 SP1.2 (WSSE)</th>
<th>Visual Studio 2005 Beta 2 (WSE 3.0)</th>
<th>Apache Axis 1.2.1 (WSS4J)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>MUST</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Must-NameSpaces</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Must-Security Headers</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Must-Signatures</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Must-Encryption</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Must-Time Stamps</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Must-Security Tokens</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MUST NOT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Must Not- Ids</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Must Not- Security Headers</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Must Not- Time Stamps</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Must Not- Encryption</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SHOULD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Should-Signatures</td>
<td>✗</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Should-Token References</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Should-Encryption</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SHOULD NOT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Should Not-Security Headers</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Should Not-Time Stamps</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

40

HS-TKT-MD-2005-011
Four legends are used to rate the tool’s performance

1. **Can’t be determined**: This legend is given in the rating when a point to be rated could not be verified at all. The common reasons for this legend are: when the infrastructure on which the analysis was carried on, is not sufficient. Second reason is the lack of scenarios to check the usage. For instance in signatures, a producer can either sign and then encrypt the signature or first encrypt the content and then digitally sign the encrypted content. There are some mandatory issues in both the type of implementations. The second case is not prevalent and scenarios where this format is followed could not be determined. Third situation was due to lack of access to proprietary and copyright issues like the XML schemas and parsers followed by the tools. In situations like these this legend is used.

2. **Does not support**: If at all sufficient evidence is very clearly available, like the vendors themselves accepting the lack of support, etc then this legend is used.

3. **Partially supports**: Every case (i.e. Must, Must Not, Should, Should Not) is further classified into codes as written in the previous section. Sometimes there are multiple mandatory constraints imposed, under each code like signatures. When it is found that all the constraints are not convincingly or clearly satisfied this legend is used. The notions of suspicions or doubts aroused behind evident support are expressed explicitly in the reasoning itself.

4. **yes completely supports**: When comprehensive support is clearly evident this legend is assigned to the code.

5. **Some points cannot be determined and some are partially supported**.

6. **Some points are supported and checked while others could not be checked**

The legends 5 and 6 are rarely used and refer to Appendix A for the contexts and reasons for there usage. Some things might be noted from the above table. One could see different patterns in the results presented, a possible explanation to these patterns is also attempted and is provided in the next section.

### 6.2 Analysis

From the Table 3 above, one can notice some common trends. Trends like all the Must cases being supported by all the tools taken for evaluation. The must cases involve checking whether URI for WS Security 2004 document is being used or not, checking for Username tokens and Binary security tokens, checking whether “must true” options give error outputs when tested against the conditions and so forth. The support for these issues may be attributed to the fact that these things involve updating the XML schema being used by the web tools as per the standards specified, which might not be a serious issue concerning Web Services tool vendors. One can also notice that Must Not - Security Headers and Must Not - Encryption cannot be verified. These involve situations where multiple hops are involved and due to the lack of infrastructure these cases are not evaluated. Considering the case of Must Not - Time Stamps, the situation with BEA where checking for clock precisions not being less than
millisecond could not be established as the schema being followed by the BEA XML parser is a proprietary issue but as per the footnote 35, timestamps are generated using a function called GENERATE_TIMESTAMP which in turn uses CLOCK_PRECISION which by default is in milliseconds. The same case was verified in Apache through the reference in the footnote where the code used for Apache engine is referenced and from certain lines in the code this checking is done.

Referring closely to the SHOULD and SHOULD NOT cases, Should Signatures case has two cases to be verified. There is an interesting point to note here one can encrypt the message and then sign the message or can sign and then encrypt the message. BEA vendor site express inability to provide encryption after signature format, refer to the Appendix A for the links but there is a provision for signature after encryption format. This case cannot be verified clearly because some specifications are not commonly used in day-to-day scenarios. That could possibly be one of the reasons why BEA even does not provide former method (encryption after signature format) of signatures. The lack of situations or cases where a need for such behaviour is needed could not be found during the evaluation phase and architecting a service to meet such a scenario is beyond the scope of the work taken in this project. “Should Token References” and “Should Encryption” codes again deal with issues that can be achieved through restructuring XML schema hence all tools stand up to the required standards. “Should Not Timestamps” again brings variation in results. The code deals with requesters and receivers relying on other applications supporting time resolution finer than milliseconds. This issue for Microsoft’s Internet Information Server could not be checked as it is a proprietary issue, the details of which are not known, but for Java it was verified. As BEA tool’s code itself is tool built (i.e. through GUI interface) there is no provision for requester or receivers to generate a time precision less than milliseconds. Coming to Apache Axis from the link provided in the footnote 39 there is no provision provided for either of the parties to generate a timestamp less than milliseconds. Here, it is also interesting to note that the BEA does not support any other types of tokens except Username tokens and Binary security tokens. The reasons behind this lack of support are not known, but can be due to the fact that other tokens like XML tokens, etc are not made mandatory by the standardization committee and one can avoid them if one does not need XML tokens in his/her application.
7 Conclusion

Lakshminarayanan [20] also presented an analogous study. It differed from our work in issues like the work carried out was only restricted to ASP.NET platform. The study strongly supports Microsoft for its contribution to the Web Services security and the conclusion reached in that case study were found to go hand in hand with the conclusions reached in our document that there was very fair support from .Net environment for WS-Security. This work is focused on the current and future standards of XML Security and describes how they are being implemented in ASP.NET and .NET Framework. Joseffson [15] came to the same conclusion that there is adequate support for WS-Security from the Web Services tools which was carried on tools provided by the software vendors like BEA, IBM, MS, and SUN. The study covered by him was in Swedish language and it was analysis of documentation provided but it did not involve implementation of the code and generation of SOAP messages, etc as carried out in our work. Secondly, there are only two tools in common which were used in both the studies and those tools are from vendors BEA and Microsoft. It is also worth to be noted that Microsoft released WSE 3.0 framework for Web Services and a new development platform Visual Studio 2005 beta 2 versions after the Joseffson’s work was done. The current study is carried on Visual Studio 2005 beta version only. It is expected from the reader to understand that the release of the specification in 2004 and the area is relatively new. So this part of the work does not have many references.

7.1 Summary

The work done here is to verify the support provided by various tools for Web Services. From the observation made, it seems evident that there is adequate and reliable support from the Web Services tools to WSS-2004. One or two issues in cases of tools like BEA, where failures are witnessed might also be considered because overall security of the entire service provided is not disturbed by the lack of availability of these features. Yet one must bear in mind that SOAP Message Security is half of the Web Services roadmap and there is a lot more work left before the complete work in Figure 9 is realized. The analysis done could outline new key issues (replacement of xerces parser by crimson in Apache Axis server, inability to reduce time precisions from milliseconds and leap seconds, etc.) should be discussed in the future. Most of the tools almost seem to stand up to the required security needs. If a tool fails to support these particular standards it can be considered as acceptable but if a tool fails to support a major and a critical constraint it must be considered severe. Inappropriate implementation of standards or appropriate implementation of standards which fail to provide real time message security are no different and to investigate situations like this would be a good work to be taken up.

7.2 Contribution

The work done can contribute to the research community to analyze the reasons behind the tools failing to rise up to the required standards. It can be the architecture followed by the tool to provide the Web Services or may be some other reason. In a comparative study, if one tool out performs other, and then a study of the underlying architecture of the successful tool can make way to the development of designing better software products.
This work also contributes to the problems highlighted in the Section 1.10 which include doubts regarding adoption of new security trends Newcomer and Lomow\textsuperscript{28} [21] in current Web Services scenario and work done regarding concerns raised by O’Neill [24] and Yang\textsuperscript{29} [39] who express doubts regarding SOAP messages in Web Services bypassing of firewalls, as a key point which would hinder the adoption of WSS-2004 by the web tools.

7.3 Future Work

It could be a good future work to verify the use of the study made here. That is to know whether this can be applied to scenarios which needed this comparative analysis or not. If it cannot, the reasons behind it and if it can, then how effective are the results drawn in decision making. A good future work can also be to verify other standards released by OASIS in these tools as this would erase all the doubts about new security threats in Web Services Security.

\textsuperscript{28} Refer to chapter 8 “Core Concepts” and “challenges to adoption”

\textsuperscript{29} Refer to page number 20 of the volume, right hand side first paragraph.
References


http://scholar.google.com/scholar?hl=en&lr=&q=cache:a04rzRbH3GgI:www.cs.up.ac.za/courses/SEC781/Assignment/References/Laksh%2520%2520ASP%2520NET%2520sec%2520WS.pdf+security+standards+in+bea


Appendix A: Rating Reasons

Microsoft Visual studio 2005 beta 2

Must

Must-NameSpaces
Referring to the MS Visual Studio SOAP Listing 1, it could be seen that the namespaces requirement imposed on the tool is satisfied.

Must-Security Headers
When checked against `<wsse:Security SOAP:mustUnderstand="1">` condition the Internet Explorer responded with a fault message hence this constraint was considered to be accepted. Nothing else could be done to check the scenario as the Microsoft is not open software and does not reveal implementation aspects due to copyrights.

Must-Signatures
Referring to the MS Visual Studio SOAP Listing 1, it can be seen from the listing that the XML Signature specification is prepended to the existing content of the `<wsse:Security>` header block.

Must-Encryption
This consists of two parts:
1. The encrypted element is prepended to security header block this could be seen from the MS Visual Studio SOAP Listing 1
2. “The sub-element MUST contain the information necessary for the recipient to identify the portions of the message that it is able to decrypt”, is the standard required and this could be verified from the MS Visual Studio SOAP Listing 1. There is information stored in the reference list about the encrypted element which allows the client to decrypt.

Must-Time Stamps
This is understood implicitly as Internet Information Server is implicitly based on UTC refer to online documentation for further information. This can be also known through the fact that when setting time time-zones are used. From the lines below

```
<wsu:Created>2004-05-28T21:16:02Z</wsu:Created>
```

one could see ‘Z’ which means zulu, an another way to refer to UTC. For more information refer to footnote.30

Must-Security Tokens
Referring to the MS Visual Studio SOAP Listing 1, it is evitable that the both username tokens and Binary security tokens are implemented.

Must Not

Must Not- Ids
Timestamps generated are machine generated and are unique. The ids are machine generated and much of the details of methods and algorithms used are not known as these are proprietary issues and have copyrights. As far the sample implementations were done id’s used such as one shown below in Must Not- Time Stamps are unique. Refer to the listing to verify this.

Must Not- Time Stamps
The usage of timestamps and their format can be clearly seen from the MS Visual Studio SOAP Listing 1.

---

30 http://www.dxing.com/utcgmt.htm links that has details regarding UTC etc.
Must Not- Encryption
This contains situations of multiple hops and such scenarios cannot be determined due to lack of infrastructure support.

Should
Should-Signatures
This actually contains points regarding signing before or after encryption. The common practice as shown MS Visual Studio SOAP Listing 1 is verified. Other case (after encryption) was not verified due to lack of scenarios but as per Foggon et.al. (2004) [9] even this can be done. Other issues such as the reference of all elements contained in the signature referring to a resource within the enclosing SOAP envelope and rejection of invalid signatures are verified and can be checked even in the listing.

Should-Token References
Referring to the MS Visual Studio SOAP Listing 1, one can see that the Token Reference is given and it has a fragment URI in a <wsse:Reference> element as per the standard demanded.

Should-Encryption
Referring to the MS Visual Studio SOAP Listing 1, one can see that all the <xenc:EncryptedData> elements created for the encryption are listed in <xenc:DataReference> elements inside the <xenc:ReferenceList> element

Should not
Should Not-Security Headers
From the snippet of the listing it can be known that enveloped transform method is employed for signatures. Refer to the SOAP listing for more information.

    <ds:Reference URI=""/>
    <ds:Transforms>
        <ds:Transform Algorithm="&enveloped;"/>
    </ds:Transform>
    <ds:Transforms>
        <ds:DigestMethod Algorithm="&digest;"/>
        <ds:DigestValue></ds:DigestValue>
    </ds:Reference>

Should Not-TimeStamps
Could not be tested as the details about IIS are not known as it is a proprietary aspect.

BEA
Must
Must-NameSpaces
The link below in the footnote[^1] refers to the available constants with BEA and the provision for the allowed namespaces. The package which provides the support is weblogic.XML.security.wsse

Must-Security Headers
When the scenario was tested for checking whether the server accepted illegal values for `mustUnderstand="true"` in security headers, the server responded with the message below.

```xml
<2005-oct-11 kl 8:24 CEST> <Warning> <WLW> <ghostserver> <cgServer>
<ExecuteThread: '13' for queue: 'weblogic.kernel.Default'> <GoodUser> <> <000000>
<Webapp-level config file
D:\Bea\user_projects\applications\SecureApp\SecureAppWeb\WEB-INF\wlw-config.XML
failed to validate against schema>
Thus, it is as per the standard demanded.
```

Must-Signatures
Referring to the Bea Listing 1, It can be found that the XML Signature specification is prepended to the existing content of the `<wsse:Security>` header block.

Must-Encryption
Referring to the Bea Listing 1, an element or element content to be encrypted by this encryption step is replaced by a corresponding `<xenc:EncryptedData>` and that is what this constraint demands.

Must-Time Stamps
The reference in the footnote\(^{32}\) below states that clearly *username token profiles and Binary token Profiles* are not implemented in BEA version 8.12. The reason for not implementing these details is not known clearly.

Must-Security Tokens
Refer to the Bea Listing 1 and it could be witnessed that both username tokens and Binary Encrypted tokens.

**Must Not**

Must Not- Ids
The ids are machine generated and much of the details of methods and algorithms used are not known as these are proprietary issues and have copyrights. As far the sample implementations were done id’s used such as one shown below are unique.

```
wsu:Id="Id-molvHaqSwynOrd4SpjaBKdFS"
```

**Must Not- Security Headers**
This cannot be determined because it involves situations with multiple hops and due to lack of infrastructure multiple hops issues cannot be determined.

**Must Not- Time Stamps**
Could be determined as it was a proprietary issue, but as per the document of the Timestamps are generated using `GENERATE_TIMESTAMP` which in turn uses `CLOCK_PRECISION`.

\(^{32}\) [http://e-docs.bea.com/wls/docs81/webserv/security.html#message_level_security](http://e-docs.bea.com/wls/docs81/webserv/security.html#message_level_security)
which by default is in milliseconds. Refer to the link in the footnote \textsuperscript{35} below for more information on this issue.

**Must Not- Encryption**
This contains situations of multiple hops and such scenarios cannot be determined due to lack of infrastructure support from the implementers side (me), reader is requested to cope with the liability.

**Should**
**Should-Signatures**
This actually contains points regarding signing before or after encryption the common practice as shown Java SOAP Listing 1 is verified. Other case (i.e. encryption after the signing) was not verified due to lack of scenarios where it is demanded. Other issues like the reference of all elements contained in the signature referring to a resource within the enclosing SOAP envelope and rejection of invalid signatures are verified and can be checked even in the listing.
But the link \textsuperscript{34} in footnotes below states that “A message-secured Web Logic Web Service first signs and then encrypts the out-going SOAP response. You cannot change this order.” So a partially supports rating is given.

**Should-Security Tokens**
From the part of listing below, reader could see that a fragment URI in ids is used to reference the tokens.

```xml
<wss:BinarySecurityToken
ValueType="wss:X509v3" EncodingType="wss:Base64Binary"
wsu:Id="Id-zPMylqRFyhQ948HEXkvY14Cx">
... <disig:Reference URI="#Id-zPMylqRFyhQ948HEXkvY14Cx">
```

**Should-Encryption**
Referring to the Bea Listing 1, one could see that all the `<xenc::EncryptedData>` elements created by this encryption step are listed in the `<xenc::ReferenceList>` element inside the sub-element. This `<xenc::ReferenceList>` element has a manifest, in order for the recipient to know the portions to be decrypted with this key

**Should Not**
**Should Not-Security Headers**
Referred to the snippet of the listing below one can see that enveloped signature transform method is followed for signatures.


\textsuperscript{34} [http://e-docs.bea.com/wls/docs81/webserv/security.html#1082126](http://e-docs.bea.com/wls/docs81/webserv/security.html#1082126)
<dsig:SignatureMethod
Algorithm="/http://www.w3.org/2000/09/XMLdsig#rsa-sha1"/>
<dsig:Reference
URI="#Id-8bcd1duqOVia2Z/1jsb0CqYB"
<dsig:Transforms>
<dsig:Transformer Algorithm="
http://www.w3.org/2001/10/XMLSchema-cl4n#"/>

Should Not-TimeStamps
Timestamps are generated using GENERATE_TIMESTAMP which in turn uses
CLOCK_PRECISION which by default is in milliseconds. Refer to the link in the footnote 35
below for more information on this issue. Both the client and server side codes are tool
generated so the question of either the server or the client requesting for a timestamp with a
precision does not exist.

Apache Axis (Java)

Must
Must-NameSpaces
Referring to the Java SOAP Listing2; it can be verified that the namespace being used is as
per the format demanded.

Must-Signatures
Referring to the Java SOAP Listing2; it can be seen from the listing that the XML Signature
is prepended to the existing content of the <wss:Security> header block.

Must-Encryption
This consists of two parts:
1. The encrypted element is prepended to security header block. This could be seen from the
Java SOAP Listing 1.
2. “The sub-element MUST contain the information necessary for the recipient to identify
the portions of the message that it is able to decrypt”, is the standard required and this
could be verified from the Java SOAP Listing 1 there is information stored in the
reference list about the encrypted element which allows the client to decrypt.

Must-Security Headers
The code responded with Null pointer exception when tested for false condition when
mustUnderstand="true" is imposed. Hence it was considered to be accepted as it primarily
based on String class and null pointer exception encountered was also due to string class.

Must-Security Headers
From the Java SOAP Listing 1 it can be seen that both Username tokens and Binary Security
tokens are implemented.

Must-Time Stamps
This could not be properly determined due to inadequate documentation in the code. But,
referring to the first link 36 in the footnotes which contains code for Timestamps, line 128
suggests that the default format is UTC and second link 37 in the footnote which consists the

35 http://e-
docs.bea.com/wls/docs90/javadoc/weblogic/XML/security/specs/TimestampConfig.html
code for **WSSecurityEngine**, lines 921 to 931 suggests that the time zone values are taken and not corrected times.

**Must Not**

**Must Not- Ids**
Id’s are usually set by the command `setID` (java.lang.Object id) and this returns two types of exceptions

`java.lang.IllegalArgumentException` - if the specified ID is not the of the correct type for this resource

`java.lang.IllegalStateException` - if this resource's ID has already been set.

Refer to link 38 in the footnotes below for more information on this topic. The second exception is raised if the demanded constraint of unique id is not used which was found when verified.

**Must Not- Security Headers**
This cannot be determined because it involves situations with multiple hops and due to lack of infrastructure multiple hops issues cannot be determined.

**Must Not- Time Stamps**
Referring to the site 39 in the footnotes below which shows the timestamps java code and referring to the line 123 one can verify that `wssConfig.isPrecisionInMilliseconds` checks even this attribute.

**Must Not- Encryption**
This contains situations of multiple hops and such scenarios cannot be determined due to lack of infrastructure support from the implementer’s side (me), reader is requested to cope with the liability.

**Should**

**Should-Signatures**
This actually contains points regarding signing **before or after encryption** the common practice as shown Java SOAP Listing 2 is verified. Other case (i.e. encryption after the signing) was not verified due to lack of scenarios where it is demanded. Other issues such as the reference of all elements contained in the signature referring to a resource within the enclosing SOAP envelope and rejection of invalid signatures are verified and can be checked even in the listing.

**Should-Token References**
One could see URI (like `<wsse:Reference URI="#7654">`) being used in the java listing 2 which is an evidence that there is a provision for this feature.

**Should Not**

**Should Not-TimeStamps**
According to the Timestamp code at the site below and referring to the lines 109-157 one could see that the valid timestamps (**line 123**) uses milliseconds and there is **no** provision for a value finer than milliseconds usage hence this can be considered.

---


Should Not-Security Headers
Referring to the Java SOAP Listing 3 it could be found that enveloped transform method is only employed and thus this constraint is satisfied.
Appendix B: Listings

Microsoft
MS Visual Studio SOAP Listing 1
The SOAP message generated is shown below.

```xml
<SOAP:Envelope xmlns:SOAP="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
    xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
    <SOAP:Header>
        <wsa:MessageID>uuid:8e53bb0d-a2eb-4c7b-9918-ff9d0a1599d9</wsa:MessageID>
        <wsa:ReplyTo>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous</wsa:Address>
        </wsa:ReplyTo>
        <wsa:To>http://localhost/partiallyencrypted/service1.asmx</wsa:To>
        <wsse:Security SOAP:mustUnderstand="1">
            <wsu:Timestamp wsu:Id="Timestamp-f90df3d5-10db-447a-961a-a7638fc010c9">
                <wsu:Created>2004-05-28T21:16:02Z</wsu:Created>
            </wsu:Timestamp>
            <wsse:BinarySecurityToken ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"
                EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-SOAP-message-security-1.0#Base64Binary"
                xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
                <wsu:Id>SecurityToken-d00793c5-ec61-418a-99ad-e345a0332btt">MIIBxDCCAWbqAwIBAgIQYpjr4F0KsIFNSd3JybItzANBqkhkiG9wUjAQQFA
DAMRQwEQYDQWDEwtSb2YUEFnZWbjeTAeFwUwMzA7MDqxODQ4MTBaFwUzOTEyMzEyMzU5NTla
```

55 HS-TKT-MD-2005-011
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/XMLenc#"/>
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/XMLenc#rsa-1_5"/>
<KeyInfo xmlns="http://www.w3.org/2000/09/XmlDSig#"/>
<wsse:SecurityTokenReference />
<wsse:Reference URI="#SecurityToken-d00793c5-ec61-418a-99ad-e345a0332bffe" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
</wsse:SecurityTokenReference>
</KeyInfo>
<xenc:CipherData>
<xenc:CipherValue>01fLD9rxujkOR0aPejrTHxsSH8zvpXAJhyjVdzHM5+3DBh4esJhutwzS5Yi48ots5F4gkBlv4ioa+9MR1r7oK31NdAlc1s118QLxLc0Z/40ocHn0YaU0sb3fXULqtvrxqz+UrDanwe2VhtTxbIHlJ62BuQaaROzsUckm062I="/xenc:CipherValue"/>
</xenc:CipherData>
<xenc:ReferenceList>
<xenc:DataReference URI="#EncryptedContent-3d793117-f020-4236-a0a0-0ed545d9bf1a"/>
</xenc:ReferenceList>
</xenc:EncryptedKey>
</wsse:Security>
</SOAP:Header>
<SOAP:Body>
<HelloWorld xmlns="http://tempuri.org"/>
</UnsecureParam>This is not encrypted.</UnsecureParam>
  <xenc:EncryptedData Id="EncryptedContent-3d793117-f020-4236-a0a0-0ed545d9bf1a" Type="http://www.w3.org/2001/04/XMLSchema#Content"
    xmlns:xenc="http://www.w3.org/2001/04/XMLSchema#">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/XMLSchema#aes128-cbc"/>
    <xenc:CipherData>
      <xenc:CipherValue>FRFClqtv8xiIHsDDShdQ8jsdBSTUTd0jNQNcLFsfqbzgYU6pBu/WVY9E9+0W5oS4</xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</SecureParam>
</SOAP:Body>
</SOAP:Envelope>

Java SOAP Listing 1: Generated SOAP Content

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=http://www.w3.org/2003/05/soap-envelope
  xmlns:xenc=http://www.w3.org/2001/04/XMLSchema#
  xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <xenc:ReferenceList>
        <xenc:DataReference URI="#EncDataId-3408129"/>
      </xenc:ReferenceList>
      <wsse:UsernameToken Id="7654">
        <wsse:Username>shambhu</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0#PasswordText">
          this is a lot of flonbar
        </wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body/>
</SOAP-ENV:Envelope>
</wsse:Password>
</wsse:UsernameToken>
</SOAP-ENVELOPEn:Security>
</SOAP-ENVELOPEn:Header>
</SOAP-ENVELOPEn:Body>
<xenc:EncryptedData Id="EncDataId-3408129">
  Type="http://www.w3.org/2001/04/XMLSchema#Content">
</xenc:EncryptionMethod>
  Algorithm="/http://www.w3.org/2001/04/XMLSchema#triplesdes-cbc"/>
</wsse:SecurityTokenReference>
  XMLNs:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wse-sec-ext-1.0.xsd"
</wsse:Reference URI="#7654" Value="UsernameToken"/>
</wsse:SecurityTokenReference>
</xenc:KeyInfo>
</xenc:CipherData>
</xenc:CipherValue>
MqrzHp/7+BDwDxwzo2e2eHSa1vucU1EDGyqcBEk5kp4SWoqEyj/GQISJjUJDH4hRRvq605K6GQ0c4yJU1uhw5dqj+Vn7wvF3XMUCY8akKjckFPbK2Uf54x9eEtO/C+iaTk5slOwkd0H5ZulYf+NjIYx+XY+Oisg/15w50d1Xog7/pGqOw8V636V7gvykz6LA9KzwgzBdKZ791y0ixFE7UEKqCm0f8v2g1KHTudFjOnF08hQYcpeHEmqKN3wRRXtcoYfzy56IxSceS9H/80RuNhf414AZnse
xy2wEmac/8KbTMzYuVKitpumxW/FFXrnxH1uvjRqtwyyqPHzQMaCncHkluXrkJod7r+vVjA3QMKvbRan45jMhsJnvKAS2Z0FyL1QXEpOpB2zdu0iNByPAUTUM+Ia3xQIRLffttU86uJ5WI6mewFLUd3K41qBaJvL9Ztw=
</xenc:CipherData>
</xenc:EncryptedData>
</SOAP-ENVELOPEn:Body>
</SOAP-ENVELOPEn:Envelope>

Courtesy Bob Daly, visited on 7 October 2005.
http://www.sims.berkeley.edu/~bdaly/cde/security/WebServicesSecurityIS219.html#webservicesSecurity_XML_Sig

Example 1: XML Signature Example
<SOAPEnv:Envelope>
  <SOAPEnv:Header>
    <wsse:Security>
      <ds:Signature>
        <ds:SignatureMethod>
          <ds:CanonicalizationMethod Algorithm="/http://www.w3.org/TR/2001/REC-XML-c14n-20010315"/>
        </ds:SignatureMethod>
        <ds:DigestMethod>
          <ds:Reference URI="#Body"/>
        </ds:DigestMethod>
      </ds:Signature>
    </wsse:Security>
  </SOAPEnv:Header>
  <SOAPEnv:Body>
    </SOAPEnv:Body>
</SOAPEnv:Envelope>
"<ds:SignedInfo>
  <ds:SignatureValue>qOvM7gmRmekoo...</ds:SignatureValue>
  <ds:KeyInfo>
    <ds:KeyValue>
      <ds:RSAKeyValue>
        <ds:Modulus>
          lvk66FUmWTBWL7Y....
        </ds:Modulus>
        <ds:Exponent>AQAB</ds:Exponent>
      </ds:RSAKeyValue>
      <ds:KeyInfo>
        <ds:Signature/>
        <wsse:Security/>
        </SOAPenv:Header>
    </SOAPenv:Body>
  </SOAPenv:Body>
</SOAPenv:Envelope>
---

  <Return-address>address</Return-address>
  <To>You</To>
  <Message>msg body</Message>
  <From>
    <ds:Signature XMLns:ds="&ds;">
      <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-XML-c14n-20010315"/>
      <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/XMLdsig#rsa-sha1"/>
      <ds:Reference URI=""></ds:Reference>
      <ds:Transforms>
        <ds:Transform Algorithm="&enveloped;"/>
      </ds:Transforms>
    </ds:Signature>
  </From>
</Envelope>
The data referenced in the `<ds:Signature>` element and specified by the "URI="" attribute of the `<ds:Reference>` descendant element is the entire `<Letter>` element, including `<ds:Signature>` itself. The instruction `<ds:Transform Algorithm="&enveloped;"/>` ensures that the `<ds:Signature>` element is excluded from the signature processing.

### Java SOAP Listing 2: SOAP for Signatures and encryption

```xml
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-RNV:Header>
    <wsse:Security
      SOAP-RNV:mustUnderstand="true"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
      <ds:Signature
        xmlns:ds="http://www.w3.org/2000/09/XmlDsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod
            Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
          <ds:SignatureMethod
            Algorithm="http://www.w3.org/2000/09/XmlDsig#rsa-sha1"/>
          <ds:Reference URI="#-3307594">
            <ds:Transforms>
              <ds:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
            </ds:Transforms>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue>
          yuc7iAar4k53b+oP6AFJDaNq16fPAtbQjmqYyY1YOf0LNVwMntOohP5q
          UXVbZ1sWTqAMYKrg3sOaa4DQgbkw==
        </ds:SignatureValue>
        <ds:KeyInfo Id="KeyId-22-000-3"/>
        <wsse:SecurityTokenReference
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
          <ds:X509IssuerSerial>
            <ds:X509IssuerName>
              CN=hsms
            </ds:X509IssuerName>
            <ds:X509SerialNumber>
```
44369778256217224376784914847992022613
</ds:X509IssuerSerial>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"

<sayHello xmlns="http://jeffhanson.com/services/helloworld">
<value xmlns=""">Hello world!</value>
</sayHello>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

BEA Listings
Bea Listing 1

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Header>
<wsse:Security
xmlns:wsse="http://schemas.xmlsoap.org/soap/2002/07/secext"
SOAP-ENV:mustUnderstand="1">
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/XmlEnc#">
<xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/XmlEnc#rsa-1_5"/>
<dsig:KeyInfo xmlns:dsig="http://www.w3.org/2000/09/XmlDsig#">
<dsig:KeyName>CN=MyCompany, O=Development, OU=MyDevTeam, L=Sealand, ST=WA, C=US</dsig:KeyName>
</dsig:KeyInfo>
<xenc:CipherData>
<xenc:CipherValue>T7XFvXJvVC9E7QcpRrxFOEcpEEr9BY5FaRkfFeg2pdrmpf+
<dsig:SignatureValue>cG9hUTE4dUSnjCH287TbDbwMlpJqlgmQexd8lu6Ktx2tNBwZfIVscg
JkFkULy/ntYCGXWdW4nq5UcTw1YUUZBxq3/3YvXzCmmtb+1bDzxZSI0RXVFmAmEHiU0vj+bbb
yXVMIQ4PBB1lemnHXVXqgDKL5aYtWMCP85Eyc+O7M=/</dsig:SignatureValue>

<wsse:SecurityTokenReference>
  <wsse:Reference URI="#zPMylqRFyhQ948HEXkvY14Cx"/>
</wsse:SecurityTokenReference>
</dsig:KeyInfo>
</wsse:Signature>
<wsse:UsernameToken
wsu:Id="Id=moviHAqSevnOrd4Sp1aBKdF5">
  <wsse:Username>GoodUser</wsse:Username>
  <wsse:Password Type="wsse:PasswordText">weblogic</wsse:Password>
</wsse:UsernameToken>
</wsse:Security>
</SOAP-ENV:Header>

<SOAP-ENV:Body Id="Id=IbcldIduwOY7a2Z/17sbUCqYB">
  <xenc:EncryptedData
XMLns:xenc="http://www.w3.org/2001/04/XMLenc#"
Id="Id=+FglEUCUoIrrOwvJ3BuBw/e"
Type="http://www.w3.org/2001/04/XMLenc#Element">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/XMLenc#aes128-cbc"/>
    <xenc:CipherData
<xenc:CipherValue>plhDUmhmkmpJALDADicvYd/LFvDDtNlxxp3HGexi1DtPr7+1AWxn/vRA-
+uhDeGwo2zx+waGBNhqQC16oQVrWFXEMepzwEY5YBzS8+pLJtdlk/ijvKGeNmmdvXHUL+
i7ufChr0Rh9XMcLfnOqP9aFcaOLv4bYrE3psFB+fgwnt2ujO+
StVK/vCREfH76Hhep0CLJq/zEOM4zPD0w2KNfG+C9X2hf8a2dND216GHuHI62T7vEXUXMDds
IrEqo5KWeqK1Y56pjFCUwjuUlaXI5FF4GOM5wG8JmdBzH7fmq8=
</xenc:CipherValue>

  </xenc:CipherData>
  </xenc:EncryptedData>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
Appendix C: Codes

Microsoft C#

The core content of the code used to verify the constraints is shown below. The coding is done using C# sharp language on IIS server.

using System;
using System.Web;
using System.Web.Services;
using System.XML;
using System.XML.Schema;
using System.XML.Serialization;

public string HelloWorld(string UnsecureParam, SecureString SecureParam)
{
    return "Secured string:"
        + SecureParam.SecureData
        + " UnsecureParam:"
        + UnsecureParam;
}

public class SecureString
{
    // Set the Oasis Id that our security reference will
    // point to
    [XMLAttribute("Id", Namespace="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd")]
    public string ID;
    [XMLText]
    public string SecureData;
}

using System;
using System.Web;
using System.Web.Services;
using System.XML;
using System.XML.Schema;
using System.XML.Serialization;

localhost.Service1Wse proxy = new localhost.Service1Wse();
X509CertificateStore store =
    X509CertificateStore.CurrentUserStore(
        X509CertificateStore.OtherPeople);
store.OpenRead();
X509CertificateCollection certs = store.FindCertificateBySubjectName("CN=WSE2QuickStartServer");
X509SecurityToken tok = new X509SecurityToken(certs[0]);
proxy.RequestSOAPContext.Security.Tokens.Add(tok);
// The "#SecureParam" matches the Id attribute value
// on our secure parameter. This tells WSE to encrypt
// that particular piece
EncryptedData enc = new EncryptedData(tok, "#SecureParam");
proxy.RequestSOAPContext.Security.Elements.Add(enc);
localhost.SecureString secStr = new localhost.SecureString();
// Set the attribute to match the Ref in the
// encrypted data object. Don't include the '#'
secStr.Id = "SecureParam";
secStr.Value = "This should be encrypted.";
string ret = proxy.HelloWorld("This is not encrypted.", secStr);

This code was written with the help of the site
All the code was written in to a .asmx file and a virtual directory was created in the IIS (Internet Information Services) active directory. And the services were invoked via the localhost through Internet Explorer browser.

**BEA**

For evaluation the sample from Pareek (2004) [26] on Web Services security was used and it could be found at site link given below. The code is, tools based and hence snap shots of the exercise are shown and actual code can be seen at the site below as this is a sample taken from the site, from the courtesy of the author.
http://dev2dev.bea.com/pub/a/2004/04/sws_wlw.html
Reader if interested could check the same, as the code is available online but it takes some time to really configure the wizards, java plug-ins and browsers. The SOAP listing generated and the snap shots of code are also shown below.

![Code view of the tool showing WSSE Policy file which is enable WSSE message security.](image)
One can notice that unlike the other two (java and Microsoft) the code is not pasted here basically for two reasons.

1. The code is available at the site referenced above and even in the Rosenberg and Remy (2004) [28]

This is the login page of the view provided.

This is the view that is provided when the result is successfully displayed.

**Java**


From the courtesy of Jeff Hanson taken from the site
http://www.devx.com/Java/Article/28816/0/page/4  on October 4th 2005. the implementation was done and tested on October 4th 2005.
The package com.jeffhanson.ws.security is contributed by Jeff Hansson. Code is altered to meet the current system settings and axis server from the apache server.

package com.jeffhanson.ws.security;
import org.apache.axis.Message;
import org.apache.axis.MessageContext;
import org.apache.axis.SOAPPart;
import org.apache.axis.client.AxisClient;
import org.apache.axis.configuration.NullProvider;
import org.apache.axis.message.SOAPEnvelope;
import org.apache.axis.utils.XMLUtils;
import org.apache.ws.axis.security.util.AxisUtil;
import org.apache.ws.security.components.crypto.Crypto;
import org.apache.ws.security.components.crypto.CryptoFactory;
import org.apache.ws.security.message.WSSignedEnvelope;
import org.apache.ws.security.message.WSEncryptBody;
import org.apache.ws.security.message.WSSignedUsernameToken;
import org.apache.ws.security.message.token.SecurityTokenReference;
import org.apache.ws.security.message.tokengetReference;
import org.apache.ws.security.WSSecurityEngine;
import org.apache.ws.security.WSConstants;
import org.apache.ws.security.WSSecurity;
import org.apache.ws.security.utils.WSSecurityUtil;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

import java.io.ByteArrayInputStream;
import java.io.InputStream;
import java.io.PrintWriter;

/**
 * Enter description here.
 * @author <a href="mailto:jeff@jeffhanson.com">Jeff Hanson</a>
 * @version $Revision: 1.1 $
 * <p/>
 * <p>&lt;b&gt;Revisions:&lt;/b&gt;
 * <p/>
 * &lt;b&gt;Jul 26, 2005 &lt;hanson:&lt;/b&gt;
 * <li>&lt;li&gt; Created file.
 * </ul>
 */

public class WSSecuritySample {

    private static final String SOAPMsg =
        "<?xml version=""1.0"" encoding=""UTF-8""?>" +
        "<SOAP-ENV:Envelope" +
        " xmlns:SOAP-ENV=""http://www.w3.org/2003/05/SOAP-envelope"" +
        " xmlns:xsd=""http://www.w3.org/2001/XMLSchema"" +
        " xmlns:xsi=""http://www.w3.org/2001/XMLSchema-instance"">" +
        "  <SOAP-ENV:Body" +

76  HS-TKT-MD-2005-011
<sayHello
XMLns="http://jeffhanson.com/services/helloworld">" +
  <value XMLns="">Hello world!</value>" +
"</sayHello>" +
"</SOAP-ENV:Body>" +
"</SOAP-ENV:Envelope>";

// The following initializes the security engine to the
// default WS-Security settings:
private static final WSSecurityEngine secEngine =
    new WSSecurityEngine();

// The following creates a crypto provider according to the
// class name specified by the system property:
    org.apache.ws.security.crypto.provider
//
// If the provider property is not set, the default class,
// org.apache.ws.security.components.crypto.BouncyCastle, is
// used.
//
// The provider is initialized to the values specified in
// the crypto.properties file. The crypto.properties file
// found in the ws4j jar file specifies
// org.apache.ws.security.components.crypto.Merlin
// as the provider class.
private static final Crypto crypto =
    CryptoFactory.getInstance();

private AxisClient engine = null;
private MessageContext msgContext = null;

/**
 * Main method
 */
public static void main(String[] args){
    try{
        WSSEnvelopeSample app = new WSSEnvelopeSample();
        Message axisMessage = app.getAxisMessage(SOAPMsg);
        SOAPEnvelope unsignedEnvelope = axisMessage.getSOAPEnvelope();
        System.out.println("<<<<<< Unsigned and Unencrypted >>>>>>")
        XMLUtils.PrettyElementToWriter(unsignedEnvelope.getAsDOM(),
                       new PrintWriter(System.out));
/*
    Message samlMsg = app.addUserTokens(unsignedEnvelope);
    System.out.println("\n<<<<<< User Tokens >>>>>>")
    XMLUtils.PrettyElementToWriter(samlMsg.getSOAPEnvelope().getAsDOM(),
                       new PrintWriter(System.out));
    Message encryptedMsg = app.encryptSOAPEnvelope(unsignedEnvelope,
                       axisMessage);
    System.out.println("\n<<<<<< Encrypted >>>>>>")
*/
}
XMLUtils.PrettyElementToWriter(encryptedMsg.getSOAPEnvelope().getAsDOM(),
    new PrintWriter(System.out));

    Message signedMsg = app.signSOAPEnvelope(unsignedEnvelope);
    System.out.println("<<<<<< Signed >>>>>");

XMLUtils.PrettyElementToWriter(signedMsg.getSOAPEnvelope().getAsDOM(),
    new PrintWriter(System.out));

    catch (Exception e)
    {
        e.printStackTrace();
    }

/**
 * WSSecuritySample constructor
 */
public WSSecuritySample()
{
    engine = new AxisClient(new NullProvider());
    msgContext = new MessageContext(engine);
}

/**
 * Creates and returns an Axis message from a
 * SOAP envelope string.
 *
 * @param unsignedEnvelope   a string containing a SOAP
 *                           envelope
 * @return <code>Message</code>   the Axis message
 */
private Message getAxisMessage(String unsignedEnvelope)
{
    InputStream inStream =
    new ByteArrayInputStream(unsignedEnvelope.getBytes());
    Message axisMessage = new Message(inStream);
    axisMessage.setMessageContext(msgContext);
    return axisMessage;
}

/**
 * Creates a signed SOAP message in compliance with WS-Security.
 *
 * @return <code>Message</code>   the signed SOAP envelope
 * as an Axis message
 * @throws Exception on error
 */
public Message signSOAPEnvelope(SOAPEnvelope unsignedEnvelope)
    throws Exception
{
    // WSSignEnvelope signs a SOAP envelope according to the
    // WS Specification (X509 profile) and adds the signature data
    // to the envelope.
    WSSignEnvelope signer = new WSSignEnvelope();

69       HS-TKT-MD-2005-011
String alias = "16c73ab6-b892-458f-abf5-2f875f74882e";
String password = "security";
signer.setUserInfo(alias, password);

Document doc = unsignedEnvelope.getAsDocument();

// The "build" method, creates the signed SOAP envelope.
// It takes a SOAP Envelope as a W3C Document and adds
// a WSS Signature header to it. The signed elements
// depend on the signature parts that are specified by
// the WSBaseMessage.setParts(java.util.Vector parts)
// method. By default, SOAP Body is signed.
// The "crypto" parameter is the object that implements
// access to the keystore and handling of certificates.
// A default implementation is included:
// org.apache.ws.security.components.crypto.Merlin
Document signedDoc = signer.build(doc, crypto);

// Convert the signed document into a SOAP message.
Message signedSOAPMsg =
(org.apache.axis.Message)AxisUtil.toSOAPMessage(signedDoc);

return signedSOAPMsg;
}

/**
 * Adds user tokens to a SOAP envelope in compliance with WS-Security.
 * @return <code>Message</code> the signed SOAP envelope
 * as an Axis message
 * @throws Exception on error
 */
public Message addUserTokens(SOAPEnvelope unsignedEnvelope)
    throws Exception {
    WSEncryptBody wsEncrypt = new WSEncryptBody();

    // Get the message as document
    Document doc = unsignedEnvelope.getAsDocument();

    String username = "joedoe";
    String password = "this is a lot ot toooor ";
    byte[] key = password.getBytes();

    // Add the UserNameToken.
    WSSAddUsernameToken builder =
        new WSSAddUsernameToken("", false);
    builder.setPasswordType(WSConstants.PASSWORD_TEXT);
    builder.build(doc, username, password);

    // Add an Id to it.
    Element usrEle =
        (Element)doc.getElementsByTagNameNS(WSConstants.WSSE_NS,
            "UsernameToken").item(0);

    String idValue = "7654";
    usrEle.setAttribute("Id", idValue);
// Create a Reference to the UserNameToken.
Reference ref = new Reference(WSSoapUtility.getDefault(WSSoapUtility()),
doc);
ref.setURI("#" + idValue);
ref.setValueType("UserNameToken");
SecurityTokenReference secRet =
new SecurityTokenReference(WSSoapUtility.getDefault(WSSoapUtility()),
doc);
secRet.setReference(ref);

// adding the namespace
WSSecurityUtility.setNamespace(secRet.getElement(),
   WSSecurity.WSSE_NS,
   WSSecurity.WSSE_PREFIX);

// Setting necessary parameters in WSEncryptBody.
wsEncrypt.setKeyIdentifierType(WSSecurity EINVAL_SECURITY_TOKEN_REF);
wsEncrypt.setSecurityTokenReference(secRet);
wsEncrypt.setKey(key);

// Encrypt using the using the key
Document encDoc = wsEncrypt.build(doc, crypto);

// Convert the document into a SOAP message.
Message signedMsg =
   (Message)Axis2Util.toSOAPMessage(encDoc);

return signedMsg;

/**
 * Encrypts a SOAP envelope in compliance with WS-Security.
 *
 * @return <code>Message</code> the signed SOAP envelope
 * as an Axis message
 * @throws Exception on error
 */
public Message encryptSOAPEnvelope(SOAPEnvelope unsignedEnvelope,
   Message axisMessage)
   throws Exception
{
   WSEncryptBody encrypt = new WSEncryptBody();
   encrypt.setUserInfo("16c73ab6-b997-458e-abf5-2f875f74882e");

   // Before Encryption
   Document doc = unsignedEnvelope.getDocument();
   Document encryptedDoc = encrypt.build(doc, crypto);

   // Convert the document into a SOAP message.
   Message encryptedMsg = (Message)
      Axis2Util.toSOAPMessage(encryptedDoc);
   String SOAPPart = encryptedMsg.getSOAPPartAsString();
   ((SOAPPart)axisMessage.getSOAPPart()).setCurrentMessage(SOAPPart,
   SOAPPart.FORM STRING);

   encryptedDoc = axisMessage.getSOAPEnvelope().getDocument();
// Convert the document into a SOAP message.
Message encryptedSOAPMsg =
    (Message)AxisUtil.toSOAPMessage(encryptedDoc);
return encryptedSOAPMsg;
}

-----method to add User Tokens

public Message addUserTokens(SOAPEnvelope unsignedEnvelope)
    throws Exception
{
    WSEncryptBody wsEncrypt = new WSEncryptBody();

    // Get the message as document
    Document doc = unsignedEnvelope.getAsDocument();

    String username = "shambhu";
    String password = "this is a lot of foobar ";
    byte[] key = password.getBytes();

    // Add the UsernameToken
    WSSAddUsernameTokenBuilder =
        new WSSAddUsernameToken("", false);
    builder.setPasswordType(WSConstants.PASSWORD_TEXT);
    builder.build(doc, username, password);

    // Add an Id
    Element userIdElement =
        (Element)(doc.getElementsByTagNameNS(
            WSConstants.WSSE_NS, "UsernameToken").item(0));
    String idValue = "/654";
    userIdElement.setAttribute("Id", idValue);

    // Create a Reference to the UserNameToken
    Reference ref =
        new Reference(WSConfig.getDefaultWSConfig(), doc);
    ref.setURI("#" + idValue);
    ref.setValueType("UsernameToken");
    SecurityTokenReference secRef =
        new SecurityTokenReference(WSConfig.getDefaultWSConfig(),
            doc);
    secRef.setReference(ref);

    // Add the namespace
    WSSecurityUtil.setNamespace(secRef.createElement(),
        WSConstants.WSSE_NS, WSConstants.WSSE_PREFIX);

    // Set appropriate parameters for key creation
    wsEncrypt.
    EMBED SECURITY TOKEN REF);
wsEncrypt.setSecurityTokenReference(secRef);
wsEncrypt.setKey(key);

// Encrypt using the using the key
Document encDoc = wsEncrypt.build(doc, crypto);

// Convert the document into a SOAP message
Message signedMsg =
    (Message)AxisUtil.toSOAPMessage(encDoc);

// Return the signed Axis message
return signedMsg;

--- Method for Signatures.

public Message signSOAPEnvelope(SOAPEnvelope
    unsignedEnvelope) throws Exception
{
    WSSignEnvelope signer = new WSSignEnvelope();

    String alias = "16c73ab6-b892-458f-abf5-2f875f74882e";
    String password = "security";
    signer.setUserInto(alias, password);

    Document doc = unsignedEnvelope.getDocument();
    Document signedDoc = signer.build(doc, crypto);

    // Convert the signed document into a SOAP message.
    Message signedSOAPMsg =
        (org.apache.axis.Message)AxisUtil.toSOAPMessage(signedDoc);

    return signedSOAPMsg;
}
Appendix D: System Information

OS Name: Microsoft Windows XP Professional
Version: 5.1.2600 Service Pack 2 Build 2600
OS Manufacturer: Microsoft Corporation
System Name: HIS-89SDSA5BDDF
System Manufacturer: Dell Computer Corporation
System Model: OptiPlex GX1 600MTbr+
System Type: X86-based PC
Processor: x86 Family 6 Model 7 Stepping 3 GenuineIntel ~598 Mhz
BIOS Version/Date: Dell Computer Corporation A08, 2000-01-06
SMBIOS Version: 2.2
Windows Directory: C:\WINDOWS
System Directory: C:\WINDOWS\system32
Boot Device: \Device\HarddiskVolume1
Locale: Sweden
Hardware Abstraction Layer Version = "5.1.2600.2180 (xpssp_sp2_rtm.040803-2158)"
User Name: HIS-89SDSA5BDDF\Administrator
Time Zone: W. Europe Standard Time
Total Physical Memory: 128.00 MB
Available Physical Memory: 17.84 MB
Total Virtual Memory: 2.00 GB
Available Virtual Memory: 1.96 GB
Page File Space: 429.38 MB
Page File: C:\pagefile.sys