PARTICIPANT DOMAIN NAME TOKEN PROFILE FOR SECURITY ENHANCEMENTS SUPPORTING SERVICE ORIENTED ARCHITECTURE

By

CHI PO CHEONG

Submitted For The Degree Of
Doctor of Philosophy

DEPARTMENT OF ENGINEERING AND DESIGN
SCHOOL OF ENGINEERING AND INFORMATICS
UNIVERSITY OF SUSSEX
BRIGHTON
UK

July 2014

DECLARATION

I hereby declare that this thesis has not been and will not be, submitted in whole or in part to another University for the award of any other degree.

Signature: _____

Chi Po Cheong

UNIVERSITY OF SUSSEX

CHI PO CHEONG – PHD ENGINEERING

PARTICIPANT DOMAIN NAME TOKEN PROFILE FOR SECURITY ENHANCEMENTS SUPPORTING SERVICE ORIENTED ARCHITECTURE

SUMMARY

This research proposes a new secure token profile for improving the existing Web Services security standards. It provides a new authentication mechanism. This additional level of security is important for the Service-Oriented Architecture (SOA), which is an architectural style that uses a set of principles and design rules to shape interacting applications and maintain interoperability. Currently, the market push is towards SOA, which provides several advantages, for instance: integration with heterogeneous systems, services reuse, standardization of data exchange, etc. Web Services is one of the technologies to implement SOA and it can be implemented using Simple Object Access Protocol (SOAP).

A SOAP-based Web Service relies on XML for its message format and common application layer protocols for message negotiation and transmission. However, it is a security challenge when a message is transmitted over the network, especially on the Internet. The Organization for Advancement of Structured Information Standards (OASIS) announced a set of Web Services Security standards that focus on two major areas. “Who” can use the Web Service and “What” are the permissions. However, the location or domain of the message sender is not authenticated. Therefore, a new secure token profile called: Participant Domain Name Token Profile (PDNT) is created to tackle this issue.

The PDNT provides a new security feature, which the existing token profiles do not address. Location-based authentication is achieved if adopting the PDNT when using Web Services. In the performance evaluation, PDNT is demonstrated to be significantly faster than other secure token profiles. The processing overhead of using the PDNT with other secure token profiles is very small given the additional security provided. Therefore all the participants can acquire the benefits of increased security and performance at low cost.

ACKNOWLEDGEMENTS

First of all, I would like to thank Dr. Pou Wan Lei for introducing the University of Sussex and Prof. Chris Chatwin to me so that I have the chance and the opportunity to do this Doctoral research.

This thesis was made possible through the help and support of lots of people, mainly from the University of Sussex, the School of Engineering and Informatics. In particular, I would like to express the deepest appreciation to my supervisor Prof. Chris Chatwin, who has the attitude and the substance of a genius. Without his guidance and persistent help, this research is impossible. In addition, I also give my thanks to my supervisor Dr. Rupert Young for all the help, patience and advice he gave me throughout my research.

And last but not the least; I would like to thank my wife and my mother, who always give me spiritual support when I am working on my thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

3-tier – Presentation tier, business logic tier and data tier.

Axis2 – a free core engine for Web Services, which is running on Apache.

Base64 Encoding – It uses radix-64 representation to represent binary data in an ASCII string format.

BPEL – Business Process Execution Language is a standard to specify actions within business process with Web Services.

CIA – Confidentiality, Integrity and Availability.

DCOM – It stands for Distributed Component Object Model. It is proposed by Microsoft and is used for communication among software components over network.

Eclipse – a free, multi-language integrated development environment (IDE).

ESB – It stands for Enterprise Service Bus. It is used to design and implement the interaction between different software applications.

ICANN – It stands for Internet Corporation for Assigns Names and Numbers.

IPSec – Internet Protocol Security is a protocol set for security Internet Protocol.

ISP – It stands for Internet Service Provider which is a telecommunication company that provides Internet connection to users.

JAR – Java Archive is a file that uses to collect a set of Java class files and associated metadata and resources.

Java API – An Application Programming Interface is a set of functions, modules, or libraries, which are written by Java programming language.

JSP – It stands for Java Server Pages which is used for creating dynamically generated Web Page using Java programming language.

OASIS – A non-profit consortium named Organization for the Advancement of Structured Information Standards. It produces many standards for security, Cloud computing, SOA, Web Services, etc.

Ontology – A set of concepts or knowledge within a domain, which are described by a set of relationships and properties.

PKI – It stands for Public Key Infrastructure.

REST – It stands for Representation State Transfer that is an architectural style and is used to implement Web Services with well-defined HTTP methods.

RPC – It stands for Remote Procedure Call. It allows a computer program to invoke a routine or module in another computer program.

SAAJ – SOAP with attachments API for Java. It is a java library used to pars and handle SOAP-based message.

SOA – Service-Oriented Architecture is a design architecture pattern used in the software design phase.

SOAP – It stands for Simple Object Access Protocol, which is a specification for exchanged structured information in XML format for implementation of Web Services.

SSL – Secure Sockets Layer is a cryptographic protocol that provides secure communication over Internet.

TCP –It stands for Transmission Control Protocol. It is a connection-oriented protocol and requires handshaking to establish end-to-end communication. It is one of the core members the Internet Protocol Suite.

UDDI – Universal Description, Discovery and Integration. It is a self-description language for a Web Service to registry themselves on the Internet.

UDP – It stands for User Datagram Protocol and is a message-based connectionless protocol. It is one of the core members the Internet Protocol Suite.

W3C – It stands for World Wide Web Consortium.

White list – a list of IP addresses that is allowed to bypass the PDNT processing rules.

WSDL – Web Services Description Language is an interface description language written in XML format. A method of Web Services can be described and invoked by other Web Services.

WSS – It stands for Web Services Security. It is secure mechanisms to protect SOAP-based message.

WSS4J – It stands for Web Services Security for Java. It is a Java implementation to support OASIS Web Services Security specifications.

XML – It stands for Extensible Markup Language (XML) that defines a set of rules for encoding a document.

XML Schema – a document uses to descript the structure of an XML document.

CHAPTER 1

- INTRODUCTION

1.1 Background

Software architecture style is changing and is categorized in different areas because as the size of the software increases so does: the complexity of functional requirements, demand for inter-system communication, data exchange between systems and integration with heterogonous systems. Shaw & Clements (1996) requires that the architectural style is formalized into a set of design rules that identify the kinds of components and connectors that may be used to compose a system or systems, together with local or global constraints on the way the composition is done. Meier et al. (2009) indicated that the architectural styles can be organized by their key focus area which includes: communication, deployment, domain and structure. They also describe examples of architecture style, for instance, Object-Oriented architecture style, Message bus architecture style, etc. However, a large and complex system is often a combination of different types of architecture styles, for instance in building a public facing Web application. The selection of architecture style is important because it will affect the software stability, expandability to support new requirements, difficulty of application deployment and maintenance of the system, especially for Internet-based or Web-based systems.

With the development of science and technology, new software architecture styles are proposed, designed and adopted. The major evolution in networking moulds software applications into new architectures, both in software and hardware architecture. The invention of Local Area Network (LAN), Wide Area Network (WAN), IP protocol (Postel J., 1981a), TCP protocol (Postel J., 1981b), Internet, etc. changed the software architecture from standalone application to layered application. Tim Berners-Lee who is the inventor of the World Wide Web (WWW) created the HTML, which is a computer language for presenting web pages and other information in a web browser. The success of the Internet and World Wide Web once again changed the software architecture from layered application to cloud computing or distributed computing. The existing distributed computing solutions such as CORBA, Java RMI imply tight coupling between various components in a system. The required high level of coordination and shared context among business systems from different organizations makes them

unreliable for open, low-overhead ubiquitous B2B e-business (Albreshne A. et al., 2009).

1.2 Evolution of Software Architecture Style

The innovation of Remote Procedure Call (RPC) leads the software architecture style into a new generation and it can be applied into many architecture styles. The RPC enables a software application to call a software module, which runs on the same or different machines. The RPC was first described in 1976 in RFC 707 and was adopted by Xerox, Microsoft, etc. One of the examples of architecture styles is Client/Server architecture, which uses RPC technology and divides a system into two applications, one is a client that makes a request and the other is a server that handles the request, for instance, a database server will return a result set when a client application makes a request. The evolution of the Client/Server architecture style was migrated from 2-tier to a multi-tier architecture and even to cloud computing because it can provide flexible and reusable applications.

The two-tier architecture is simple but comes at the cost of lack of scalability. The business logic in two-tier architecture can be placed in either the user interface (presentation layer) or database (data tier) and the user interfaces directly access the database. However, the most commonly adopted multi-tier architecture is the three-tier architecture (3-tier). Basically the 3-tier style can be split into three tiers or layers, (1) Presentation Tier, (2) Business Logic Tier and (3) Data Tier. In order to improve reusability and flexibility of a complex system, more tiers or layers are added, for instance, a new tier or layer is added to handle business workflow. The evolution of tier architecture is shown in Figure 1.1. Each tier is independent of each other and it can be upgraded or replaced individually without modifying other tiers.

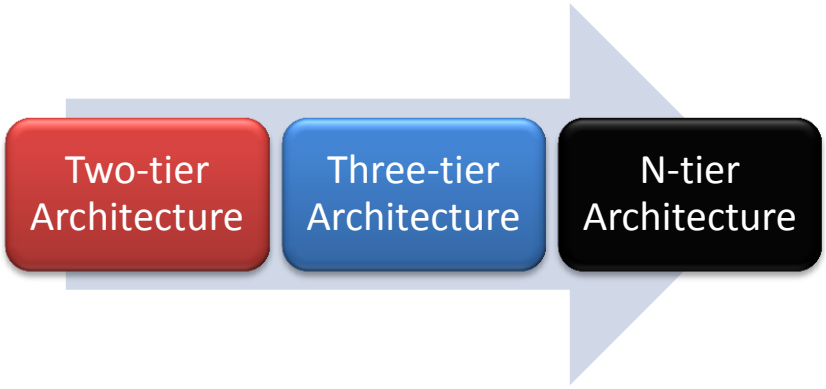

Figure 1.1: The evolution of tier architecture

Cloud computing and grid computing have become buzzwords after Web 2.0. Godfrey B. (2006) identifies that distributed computing works by splitting up the larger task into smaller chunks, which can be performed at the same time independently of each other. Grid computing is a distributed computing concept which is used for sharing and integrating computing resources, it evolved from heterogeneous systems. Grid computing aims to solve the common IT problem of dedicated and underutilized hardware resources. It is also used to divide a large task into many tasks that run in parallel on separate servers. The success of the Internet led to the concept of cloud computing (Boss G. et al., 2007). A cloud computing infrastructure can do large-scale processing and it is massively scalable. Due to the nature of the Internet, cloud computing can break down the physical barriers and link the hardware and software platforms in different global locations through the Internet. Thousands or millions of computers can create an enormous machine pool. A complex and time-consuming task running on a vast amount of computers can obtain an expected result within minutes instead of weeks or months.

Many highly complex applications have moved to the World Wide Web (WWW) platform which has caused the Internet to grow dramatically. Therefore, a new architectural style was proposed and adopted in many Web-based systems; this is the Service-Oriented Architecture (SOA). The degree to which source codes can be reused is very important and has changed software engineering as shown in Figure 1.2. Reusable modules and classes can reduce implementation time, testing time and has

eliminated application bugs. Service reuse is one of the main advantages of adopting SOA. The service not only includes the business logics, it also includes the business processes. Services can work together representing an implementation of a business process or a business flow.

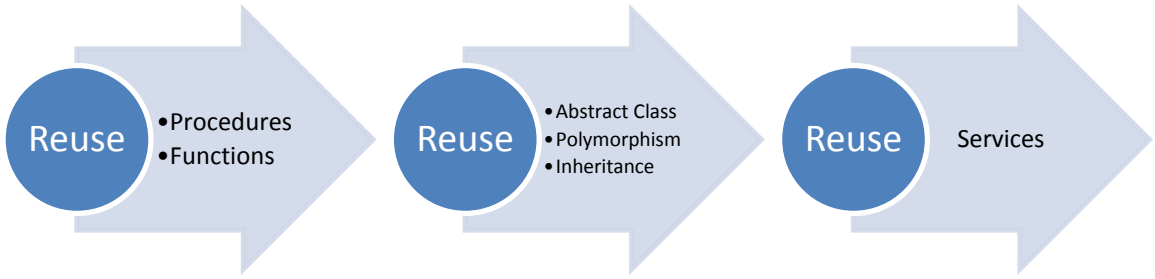

Figure 1.2: The Degree of source code reusability

The SOA has emerged over the past years as one of the preferred approaches for system design, development, and integration (Davis, 2009). SOA is a design architecture pattern used in the software design phase. An SOA-based system is a kind of distributed system that splits and groups the business logics into a loosely coupled set of services, usually called Web Services. SOA is not a technology but it is an architecture style and it can be implemented in many different ways, such as CORBA or with other Remote Procedure Call (RPC) technologies. However, Web Services is widely used to implement an SOA-based system. Web Services is a message-based communication between service provider and service consumer. Booth et al. (2004) identifies two major classes of Web services architecture as illustrated in the following:

- REST-compliant Web services, in which the primary purpose of the service is to manipulate XML representations of Web resources using a uniform set of "stateless" operations; and
- Arbitrary Web services, in which the service may expose an arbitrary set of operations.

1.3 Motivation

Simple Object Access Protocol (SOAP) is one of the methodologies to implement Service-Oriented Architecture (SOA). The SOAP protocol relies on Extensible Markup Language (XML) for its message format. A structured message is used to exchange data between service consumer and service provider in Web Services architecture. Therefore, a secure Web Services system not only focuses on the security of the system itself (e.g. hardware and software), but also on the confidentiality and Integrity of the message exchange between participants throughout the Internet, which is an unsafe public network, which is two components of the CIA (Confidentiality, Integrity and Availability) Triad in the information security field. Parker D. (2002) also proposed an alternative model, which named six atomic elements of information which include confidentiality, possession, integrity, authenticity, availability and utility.

The SOAP-based Web Services uses WS-Security 1.1 OASIS standard which is approved and published by Advancing Open Standards for the Information Society (OASIS) to fulfill the security requirements in SOAP message exchanges among participants. Moreover, the WS-Security 1.1 also provides security extensions for the Web Services protocol stack to provide end-to-end message security. The following specification and profiles make up the WS-Security 1.1 OASIS standard.

- WS-Security Core Specification 1.1
- Username Token Profile 1.1
- X.509 Token Profile 1.1.
- SAML Token Profile 1.1
- Kerberos Token Profile 1.1
- Rights Expression Language (REL) Token Profile 1.1

The WS-Security Core specification and five secure token profiles are based on different existing security mechanisms and open standards including PKI, X.509, Kerberos, or other algorithms. However, they are complex and produce a lot of overhead, especially the X.509-based encryption and signature. If many SOAP

messages are exchanged between service consumer and service provider, the overhead will be increased significantly. Moreover, as more secure token profiles are added to a SOAP message, more overheads will be produced for parsing and handling SOAP messages.

"Who" can use the services, "What" is the authorization is tackled by the WS-Security 1.1 OASIS standard. However the "Where" is not handled or supported. For instance, a user John is allowed to use or invoke a service and has been granted appropriate user rights. According to the WS-Security 1.1 OASIS standard, the user can use or invoke the services everywhere. It is a security hole if the services should only be provided for a particular enterprise or domain. Therefore, a new secure token profile is needed to solve this issue and integrated with WS-Security standard.

1.4 Objectives

Although there are several security standards currently available and adopted in a SOAP-based message, they will produce a lot of overhead if one or more secure token profiles are used in the SOAP message. In this research, the location of service consumer and provider is taken into consideration for message exchange, parsing and handling. The objectives include the following:

(i) Research on the WS-Security 1.1 OASIS standard, present the details.
(ii) Find out the limitations of WS-Security 1.1 OASIS standard.
(iii) Propose a new token profile to tackle the limitation found in Objective [ii].
(iv) Develop a parser and processor based on Objective [iii].
(v) Building a performance model to evaluate Objective [i] and Objective [iii].
(vi) Present the experimental results and analysis based on Objective [v].

1.5 Innovations

The WS-Security 1.1 OASIS standard is complex and produces a lot of overhead, especially the XML encryption, XML signature and other PKI-based mechanisms. However, the location of the service consumer and provider is not verified. It is a security hole if a service should only be provided for a particular enterprise or domain.

The newly designed secure token profile incorporates one of the most widely used Internet service information identity, the Domain Name Service (DNS). It can verify, control and monitor the location of the service consumer or message sender. The newly designed token can be used to reject invalid requests or responses, which come from unknown or fake domains before parsing and processing the WS-Security 1.1 OASIS secure token profiles. Therefore, the service provider or consumer can save processing resources and handle more valid Web Services requests or responses.

CHAPTER 2

– A LITERATURE REVIEW OF SOA, WEB SERVICES AND SECURITY

2.1 Service-Oriented Architecture (SOA) and Web Services

Service-orientation is a design paradigm comprised of a specific set of design principles. The application of these principles to the design of solution logic results in a service oriented solution logic. The most fundamental unit of service-oriented solution logic is the service (Erl T., 2007). The architecture style defining an SOA describes a set of patterns and guidelines for creating loosely coupled, standards-based business-aligned. Service-Oriented Architecture is an IT strategy that organizes the discrete functions contained in enterprise applications into interoperable, standards-based services that can be combined and reused quickly to meet business needs [BEA, 2005]. Erl T. (2005) indicated that the common tangible benefits of SOA are:

1. Improved integration
2. Inherent reuse
3. Leveraging the legacy investment
4. Establishing standardized XML data representation
5. Focused investment on communications infrastructure
6. "Best-of-breed" alternatives

Artus (2006) also mentioned that Service-Oriented Architecture (SOA) offers a version of IT flexibility enabling business agility. More and more enterprises are realizing that SOA is a reality not a myth. The information systems landscape of most enterprises is heterogeneous in nature and comprises of a combination of legacy and modern systems (Kumari et al., 2008). A Gartner survey (Sholler, 2008) showed that more than 85 percent of Fortune 1000 companies are engaged in a service-oriented initiative, and that the average SOA project has been in place for about 9 months. Another survey (TechTarget/Forrester Research, 2010) showed that 47.4% of respondents work in organizations where SOA projects are underway, and 30.9% have multiple SOA projects underway. In terms of project scope, this work is seen as "enterprise-level" in nature in 62.6% of the cases. Formal SOA offices or centers of excellence exist in 13.6% of organizations, up from 9.0% in a comparable 2009 survey. However, Lewis et al., (2007) identified eleven common misconceptions about SOA including:

1. SOA provides the complete architecture for a system
2. Legacy systems can be easily integrated into an SOA environment
3. SOA is all about standards and standards are all that is needed
4. SOA is all about technology
5. The use of standards guarantees interoperability among services in an SOA environment
6. It is easy to develop applications based on services
7. It is easy to develop services anybody can use
8. It is easy to compose services dynamically at runtime
9. Services can only be business services
10. Testing applications that use services is no different than testing any other application
11. SOA can be implemented quickly

Maamar Z. et al., (2011) indicated that Service-oriented architecture (SOA) and its flagship implementation technology known as Web services have changed the way software engineers design and develop today's enterprise applications. SOA is not a technology but it is an architecture style or architecture pattern. It is technology independent and can be implemented by different remote procedure call technologies, for instance, Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), etc. The implementation of SOA applications is made possible through the realization of Web Services (Lee et al., 2006). Mahmoud (2005) identified that Web Services are the preferred standards-based way to realize SOA. A Service-Oriented Architecture (SOA) can be implemented by a collection of loosely coupling services, which can be invoked by each other to exchange information between participants. Web Services are widely adopted and used to implement the SOA because of the maturing set of standards.

Web Services, at a basic level, can be considered a universal client/server architecture that allows disparate systems to communicate with each other without using proprietary client libraries (Judith M., 2009). Web Services systems enable a high level of decoupling as well as dynamic binding of services. Such systems are composed by services, which contain behavior and messages. Services are found by applications

using service discovery (Gottschalk K., 2000). There are many Web Services definitions described in different books or Websites, Albreshne A. et al. (2009) gives the following overview:

- A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. Because all communication is in XML, web services are not tied to any one operating system or programming language--Java can talk with Perl; Windows applications can talk with UNIX applications.
- Web Services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or Web based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.
- A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

Therefore, a Web Service is an XML-based platform-independent protocol. Moreover, it is also programming language independent, and can be implemented using different programming languages. The Web Services can be distributed in both Internet and Intranet and discovered through a simple mechanism. Nowadays, many commercial products build-in Web Services technology, for instance, Web Services in Exchange 2013 or Exchange Web Service (EWS). EWS is a SOAP-based web service API that you can use to communicate with Exchange. EWS uses HTTP POST requests to send commands and data to the service endpoint (MSDN, 2013). Amazon also provides a Web Services (AWS) infrastructure to their customers. It delivers a scalable cloud

computing platform with high availability and dependability, offering the flexibility to enable customers to build a wide range of applications. Helping to protect the confidentiality, integrity, and availability of our customers' systems and data is of the utmost importance to AWS, as is maintaining customer trust and confidence (Amazon, 2013). Fusaro VA., et al., (2011) use AWS to deliver their biomedical cloud computing.

2.2 SOA-Based System

An SOA-based system is also a kind of distributed system and similar to the Internet-based system. One of the major differences is the level of dependence on business logics in the design phase. A traditional Internet-based system decomposes the business logics into more tightly-coupled components but SOA splits it into loosely coupling components and the SOA relies on components and creation of services. The other difference is that the system architecture of a SOA-based system can be changed after deployment. Therefore, a new service can be added in the system without stopping the whole system. The potential benefits of SOA are services reuse, integration improvement, leveraging the legacy investment and best of breed integration. SOA is suitable to design a distributed, Internet-based, dynamic change, autonomous and non-point to point system.

The following sections show an example which is designed in SOA and implemented using Web Services. The example is "An SOA-based Disease Notification System" (Cheong C. P. et al., 2009). It proposes a Diseases Notification System which is designed using a SOA pattern. Disease notification is an import component of the disease control systems. A successful diseases notification system will deliver positive effects for human beings. The proposed system is designed and implemented by use of the Business Process Execution Language for the business process layer and Enterprise Service Bus for the connectivity layer. Therefore, it can cope with a dynamically changing environment or requirements. Moreover, the proposed system can reduce the notification process time and it can provide timely information on diseases notification.

2.2.1 An SOA-Based Diseases Notification System (SOADNS)

Many computer-based systems or tools are used for monitoring of disease migration. Some provide record-based information and some provide visual data, examples are given by (Chang L. C. et al., 2005) and (Qian Z. et al., 2004). A location-based data visualization system can be of great benefit for disease monitoring and tracking. However, a successful disease control system is based on cumulative recording of the occurrence of spreading diseases. Diseases notification and tracking is a vital component in ensuring protection of public health. To prevent and control the spread of infectious disease around the world, health organizations must monitor trends over time not only in human diseases such as chickenpox, tuberculosis, plague, HIV, Severe Acute Respiratory (SARS) especially, the spreading disease: Influenza-A H1N1, etc. and even animal diseases such as bird flu H5N1 and H7N9.

2.2.1.1 System Overview

The SOA-based Diseases Notification System (SOADNS) utilizes medical standards to exchange or identify notifiable diseases, including: International Classification of Diseases (ICD) and Health Level Seven (HL7). A doctor can use the ICD code to classify each disease. For instance, code “487” represents “Influenza” in ICD version 9CM and code “J11” represents “Influenza, virus not identified” in ICD version 10. The SOADNS has two roles, one is service provider and the other is service consumer. For instance, it provides services to the disease declarer and consumes service provided from the local health authority. A service consumer can discover the service by the Universal Description and Discovery and Integration (UDDI) or service brokers. Earlier SOA-based systems can be implemented using different connection technologies such as Distributed Component Object Model (DCOM) or Common Object Request Broker (CORBA). However, the SOADNS uses Web services which will be implemented by the Simple Object Access Protocol (SOAP), Web Service Definition Language (WSDL), etc. to implement the system. Figure 2.1 shows the high level architecture of the SOADNS. A clinic or a private doctor can declare notifiable diseases by use of a Web interface provided by the local health authority. The Hospital acts as a service consumer

and submits the notifiable diseases by use of a standard SOA client application using a SOAP message format. The hospital can also act as a service provider to provide Web services to other service consumers. For instance, it can provide the details of patient records to the WHO if necessary. The SOADNS is designed as a standard application package. Therefore, it can be deployed to every participant and only requires minor changes in the business process layer.

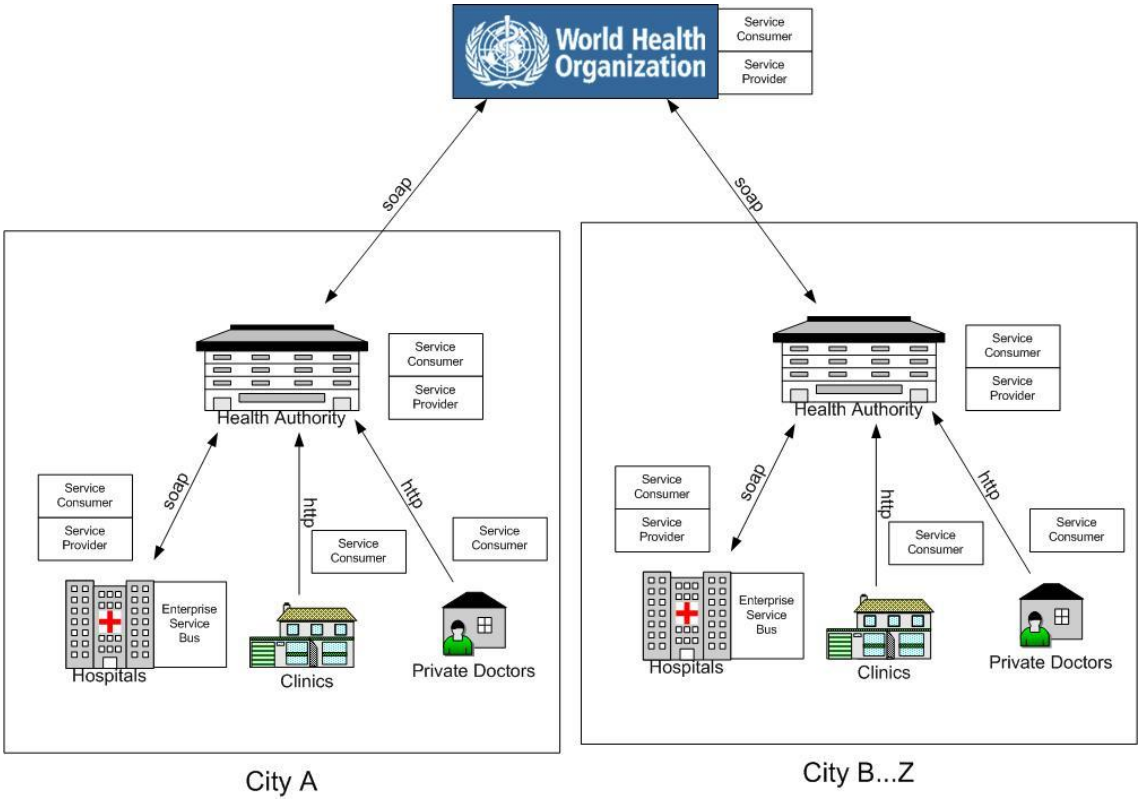

Figure 2.1: High level architecture diagram of SOADNS

2.2.1.2 System Architecture

The SOADNS is composed of five layers including: presentation layer, service layer, business process layer, connectivity layer and data layer. The presentation layer - the client side uses a Web browser to input data, such as patient records and diseases notification data. The service layer contains all the services provided from the proposed system. The service can encapsulate one or more business logics of the system and it

has an interface for the service consumer to interact with. Services are orchestrated into business processes in the business process layer. The Business Process Execution Language (BPEL) is used in this layer and it can specify which services should be invoked and the sequence in which they are invoked, which can ensure the business/work flow is in the correct order. Enterprise Service Bus (ESB) is used in the connectivity layer. It can move, transform and route the clinical data within the internal system and between the external systems in XML format. The data layer includes the database of diseases notification data, demographic data and it can be used through an adapter service defined in the connectivity layer, for instance, a database adapter service.

A. Service Layer

Service modeling is one of the vital processes to build a SOA-based System. The process decomposes the business logics into several generic processes or services. There are several core services in SOADNS including: CodeValidationService, NotificationService, TaskService and DatabaseAdapterService. The NotificationService is used for notifying guarders when a serious incident occurs via email, fax, pager or SMS. The ICD Code is validated by the CodeValidationService. It can handle the different versions of the disease codes. The TaskService is a human workflow task and it can interweave human interactions with connectivity to the SOADNS. The human task is linked to the Business Process Layer through a WSDL. In the SOADNS, the TaskService is used for the human re-approval and re-confirmed for the disease being monitored before notifying the emergency committee. DatabaseAdapterService provides a bridge between the database and the business process layer. All the services can interact with each other at the message level and are described in Web Service Orchestration language.

B. Business Process Layer

The Business Process Execution Language (BPEL) is used in this layer for specifying interactions with Web Services including the business logic and the execution order. The BPEL process of SOADNS is shown in Figure 2.2. For instance, a doctor submits

a notifiable disease through a web browser. The doctor can input different versions of ICD code. Then, the BPEL process is invoked by a message from the application server or ESB. The BPEL process writes the notifiable disease record to the database. The BPEL process calls the CodeValidationService to valid the disease code. If the disease code is valid, it will check whether it is required to notify a monitor or a committee member by using NotifialeDiseaseDBAdapter. The committee or members of the WHO may receive an alert message under certain circumstances. Finally, an email will be sent back to a declarer for confirmation. If it is a confirmed case, the system will send an alert message through SMS or a pager to the emergency committee. Otherwise, if it is a suspected case, the system will only send an alert message via an email to a general officer for example.

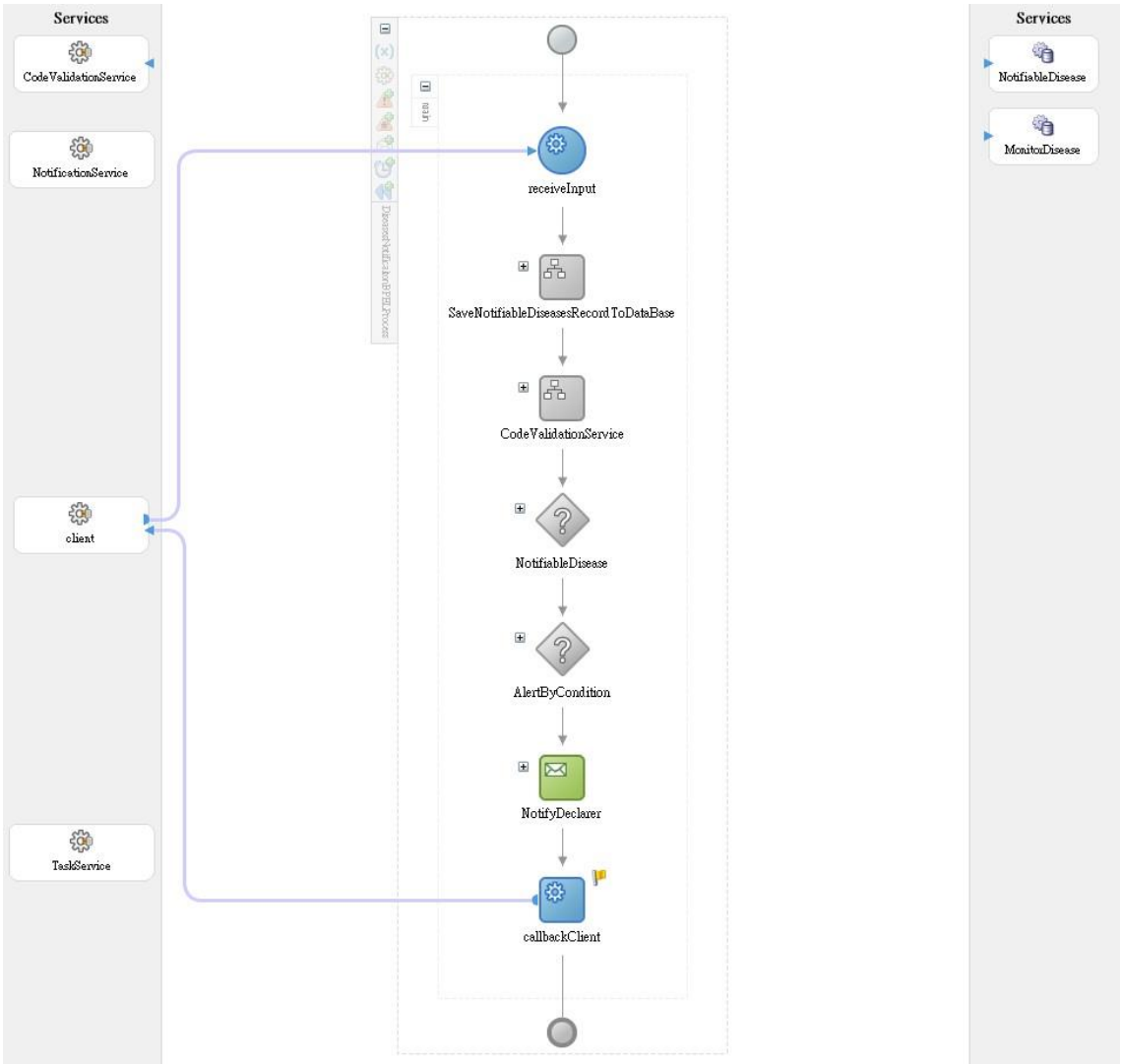

Figure 2.2: BPEL process of SOADNS

C. Connectivity Layer

In a hospital environment, many different architecture systems, which include hardware and software, are used for different purposes. For instance, a Health or Hospital Information System is used for recording patient demographic and clinical information. The Lab or laboratory management system (LMS) is used in the laboratory for the management of the patient's samples such as blood test samples. It requires an interfacing program between the LMS and laboratory equipment. Different systems produce outputs in different formats, such as text, XML, HL7, etc. Therefore, the Enterprise Service Bus (ESB) can be used to put everything together. The ESB is suitable for use in the hospital environment. It can collect the output coming from different platforms and in different formats. It means that the disease code not only comes from user input through a Web user interface, it can also be obtained from heterogeneous systems automatically.

2.2.1.3 Advantages of SOADNS

The SOADNS is designed in a SOA style. Therefore, it can obtain the advantages of SOA such as business logic reusability, loosely-coupled to other services, etc. The BPEL and ESB are also used to make the system complete. The system can be dynamically changed by modifying the BPEL process in the business process layer. Therefore, the system is easy to spread and to deploy. The ESB is used in the SOADNS. It can accommodate the data of existing medical systems, such as laboratory management system, X-Ray management system, etc. The SOADNS is composed of services, BPEL and ESB. Therefore, it increases flexibility, it is easy to change the system architecture and it scales from a point to point solution to a distributed solution. Moreover, the SOADNS can make diseases notification more effective and provide timely information. It uses BPEL as a standards-based way of orchestrating all services. The system architecture can be changed easily by modifying the business process layer through BPEL. Therefore, the SOADNS can be deployed by different participants without modifying the application source code. The ESB is used to collect notifiable diseases data in different formats from heterogeneous systems. The proposed system is a complete solution for diseases notification in and from different locations.

2.2.1.4 Security consideration of SOADNS

One of the major security issues is who can submit a notifiable disease, for instance, a private doctor, a clinic or even a hospital. Usually, a user or a participant can be identified by a username with a password. Therefore, the SOADNS can adopt the Username Token Profile, which is one of the secure token profiles collected in the OASIS standard 1.1 to fulfill the user authentication requirement. However, "where" can submit a notifiable disease is not handled and not verified in existing secure token profiles. Therefore, a new secure token profile is a need to provide additional security for use by SOANDNS.

2.3 Web Services Architecture

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format. Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards (Booth et al., 2004).

Figure 2.3: Web service architecture (Champion et al., 2002)

The Web Services architecture basically consists of three roles, service requestor, service provider and service registry as illustrated in Figure 2.3. The relationship between these three roles is: a service provider defines and publishes a service description to a service requestor or service registry; the service requestor finds a desired service and retrieves the service description locally or from the service registry; the service requestor uses service description to bind with and invoke the service implementation from the service provider directly.

2.4 Classes of Web Services

Two major classes of Web services are defined by W3C. Representation State Transfer (REST)-compliant Web services and arbitrary Web services, the "Big" Web services technology. REST is neither a standard nor a protocol. It is an architectural style like the client/server architecture. The arbitrary or "Big" Web Services is a web service which is implemented by Simple Object Access Protocol (SOAP).

2.4.1 REST-compliant Web Services

RESTful Web Services are an alternative to Remote Procedure Call technologies like SOAP and WS-* services (Schreier S., 2011). According to Fielding (2000), Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: Presented with a network of web pages (a virtual state-machine), the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use. REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages (Rodriguez, 2008). Hansen (2007) identified a set of constraints and architectural principles for the REST-style services as illustrated in the following:

- RESTful services are stateless. As Fielding (2000) writes in Section 5.1.3 of his thesis, "each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server."
- RESTful services have a uniform interface. This constraint is usually taken to mean that the only allowed operations are the HTTP operations: GET, POST, PUT, and DELETE.
- REST-based architectures are built from resources (pieces of information) that are uniquely identified by URIs. For example, in a RESTful purchasing system, each purchase order has a unique URI.
- REST components manipulate resources by exchanging representations of the resources. For example, a purchase order resource can be represented by an XML document. Within a RESTful purchasing system, a purchase order might be updated by posting an XML document containing the changed purchase order to its URI.

One of the major characteristics of the RESTful Web Services is the explicit use of well-defined HTTP methods which are defined in RFC 2616. It maps the well-known HTTP methods to four basic functions which are "create", "read", "update" and "delete" (CRUD) operations as shown in Table 2.1. In a traditional Web application, it uses the HTTP GET method with "query string" or "hidden field" which is defined in standard HTML form to achieve the four basic functions (CRUD). However, this mechanism changes the original design of HTTP protocol and it is inconsistent with the HTTP GET method. An example of an Application Program Interface (API) between RESTful Web Services and traditional Web application is shown in Table 2.2.

REST-based resource-oriented Architecture abstracts all the objects in the system as resources and gives them their own unique resource identifiers. It provides various resource services to the external environment through RESTful Web Services (Wang et al., 2009). The RESTful utilizes Uniform Resource Identifier (URI) to identify resources and uses uniform interface (HTTP methods) to manipulate information. It

uses a directory structure-like URI to make a service consumer easily understand where they can obtain the resources. Table 2.3 shows the examples of RESTful URI for acquiring particular resources. The REST-style architecture is like the client/server architecture, for instance, a client sends a request to a server and a result will be sent back to the client after the server processes the request. It is similar to a user browsing a web site on the Internet. For instance, a user browses a web site by sending an HTTP request to a web server. The web server parses and processes the HTTP request and returns a result depending on the HTTP method.

Table 2.1: The mapping between RESTful operations and the HTTP method

RESTful Operations	HTTP method	Description
Create	POST	Submit data to be processed by the identified resources
Read	GET	Request a specific representation of a resource
Update	PUT	Update a resource with the supplied representation
Delete	DELETE	Deletes the specified resource

Table 2.2: The API of RESTful Web Services and traditional web application

Operation	RESTful API	Traditional Web API
Create	POST /students HTTP/1.1 Content-Type: application/xml …... <?xml version=”1.0”?> <student>	GET /addStudent?name=Peter HTTP/1.1

	<name>Peter</name> </student>	
Read	GET/students/Peter HTTP/1.1 Accept: application/xml ……	GET / getstudnet?name=Peter HTTP/1.1
Update	PUT /students/Peter HTTP/1.1 Content-Type: application/xml …… <?xml version="1.0"?> <student> <name>David</name> </student>	GET /updatestudent?name=Peter&newname=David HTTP/1.1
Delete	DELETE/students/Peter HTTP/1.1 Accept: application/xml ……	GET /deleteStudent?name=Peter HTTP/1.1

Table 2.3: Example of RESTful URI for acquiring particular resources

URI	Description
http://www.sussex.ac.uk/students	Resources of students
http://www.sussex.ac.uk/students/1001	Resource of student ID 1001
http://www.sussex.ac.uk/schools	Resources of schools
http://www.sussex.ac.uk/schools/Informatics	Resource of school of Informatics

Due to the simplicity and flexibility, REST is suitable for a constrained environment. Many Web clients and physical devices support HTTP protocol. It can easily cooperate and integrate with different devices if security is not a major issue. In recent years, some REST security protocols (Serme G., 2012), (Kosmajac D., 2012) have been proposed in academia. However, they are not standard security protocols. SOAP-based Web

Services has a De factor Web Services security standard. Therefore, it will be adopted if security is a major concern.

2.4.2 SOAP-based Web Services

Simple Object Access Protocol is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics (Gudgin M. et al., 2007). Al-Zoubi et al. (2009) identify that the SOAP-based Web Services are provided as Remote Procedure Calls (RPC) on top of the Web (HTTP). SOAP is a protocol specification for exchanging structured information in the implementation of Web Services in a computer network. It relies on eXtensible Markup Language (XML) for its message format, and usually relies on other Application Layer protocols for message negotiation and transmission (Atadjanov A.J., 2010).

A method or a subroutine hosted in a local machine can be executed by a remote system through an Application Programming Interface (API). However, a traditional API is language-dependent and is only available for a particular programming language. A Web service is a kind of distributed system and it provides APIs which can be used by all systems on the Internet. Currently, SOAP is the most common way to implement Web Services and it is a core component of the Web Services architecture. The SOAP is a specification for exchanging structured messages between two computer systems via common communication protocols. The components of SOAP are shown in Figure 2.4. XML is used for exchanging information and HTTP, SMTP, etc are used for the communication between participants.

Figure 2.4: The components of SOAP

A valid SOAP message is a well-formed XML document, which is defined in the XML 1.0 specification. The SOAP message consists of a SOAP envelope, SOAP header and SOAP body as illustrated in Figure 2.5. The SOAP envelope element identifies the XML document as a SOAP message and indicates the start and the end of the SOAP message. The SOAP header is an optional element, which provides application-specific information for extending the functionality of a SOAP message. The SOAP body element contains the method name with corresponding parameters of the Web Services and the results after invoking the Web Services. It also contains a child element, a SOAP fault element, which is used to indicate error messages while processing the message. A SOAP message is encoded using XML and uses SOAP namespace and a SOAP encoding style as shown in Table 2.4. Mitra N. and Lafon Y. (2007) indicate how to invoke a SOAP RPC as shown in the following:

1. The address of the target SOAP node.
2. The procedure or method name.
3. The identities and values of any arguments to be passed to the procedure or method together with any output parameters and return value.
4. A clear separation of the arguments used to identify the Web resource, which is the actual target for the RPC, as compared with those that convey data or control information used for processing the call by the target resource.
5. The message exchange pattern, such as HTTP POST and HTTP GET, which will be employed to convey the RPC, together with an identification of the so-called "Web Method"
6. Optionally, data, which may be carried as a part of SOAP header blocks.

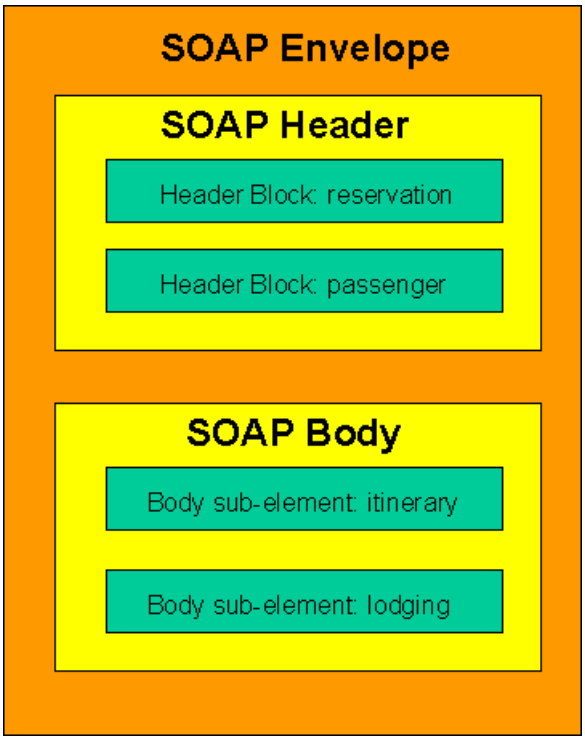

Figure 2.5: A pictorial representation of the SOAP message (Mitra N. et al., 2007)

Table 2.4: Prefixes and namespaces used in this research

Prefix	Namespace	Notes
S11	http://schemas.xmlsoap.org/soap/envelope/	A normative XML schema document for SOAP message version 1.1 namespace
S11:encodingStyle	http://schemas.xmlsoap.org/soap/encoding/	indicates the encoding rules used to serialize a SOAP message version 1.1
S12	http://www.w3.org/2003/05/soap-envelope	A normative XML schema document for SOAP message version 1.2 namespace
S12:encodingStyle	http://www.w3.org/2003/05/soap-encoding	indicate the encoding rules used to serialize a SOAP message version 1.2

Usually, the SOAP uses HTTP as a communication protocol, named SOAP HTTP binding. A SOAP message is embedded in a HTTP request or HTTP response message that complies with the SOAP encoding rules. A HTTP POST method is used for the SOAP request message and uses HTTP Content-Type header to define MIME (Multipurpose Internet Mail Extensions / Internet Media) type and character encoding. An example of a SOAP request and response message is shown in Figure 2.6 and Figure 2.7. According to the RFC 3902, the Content-Type is defined by the MIME type of the body as illustrated in Table 2.5. The Content-Length header represents the number of bytes in the request and response body. In this example, a method "GetStudents" is sent to a Web server with a parameter "studnetId" and the student information who student id is A001 will be returned in XML format. The namespace and the schema of student information are defined in the "http://www.sussex.ac.uk/student" in this example.

Table 2.5: The Content-Type used in SOAP request message

MIME media type name	Application
MIME subtype name	soap+xml
Required parameters	None
Optional parameters	charset: specified in RFC 3023 actions: used to specify the URI that identifies the intent of the message.

```
POST /student HTTP/1.1
Host: www.sussex.ac.uk
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
     S12:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

     <S12:Body xmlns:ns="http://www.sussex.ac.hk/student">
          <ns:getStudents>
               <ns:studentId>A001</ns:studentId>
          </ns:getStudents>
     </S12:Body>

</S12:Envelope>
```

Figure 2.6: An example of SOAP request message using HTTP protocol

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<S12:Envelope xmlns:S12="http://www.w3.org/2003/05/soap-envelope"
     S12:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

     <S12:Body>
          <getStudentsResponse xmlns:ns="http://www.sussex.ac.hk/student">
               <student>
                    <studentId>A001</studentId>
                    <firstName>Peter </firstName>
                    <lastName>Chan</lastName>
                    <gender>M</gender>
                    <email>peter.chan@sussex.ac.uk</email>
               </student>
          <getStudentsResponse>
     </S12:Body>
</S12:Envelope>
```

Figure 2.7: An example of SOAP Response message using HTTP protocol

The Web Services Description Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information (Christense E., et al., 2001). A SOAP client uses WSDL to locate Web Services and obtain a method description as illustrated in Figure 2.8. Therefore, the client can know how to access the services. It also indicates which transport protocol is used, for instance, the HTTP protocol.

```
<definitions name="StudentService"
      targetNamespace="http://www.sussex.ac.uk/wsdl/StudentService.wsdl"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:S11="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:tns="http://www.sussex.ac.uk/wsdl/StudentService.wsdl"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:s0="http://www.ssussex.ac.uk/student/student.xsd>

      <message name="GetStudentRequest">
            <part name="studentId" type="xsd:string"/>
      </message>
      <message name="GetStudentResponse">
            <part name="student" type="s0:Student"/>
      </message>

      <portType name="Student_PortType">
            <operation name="getStudents">
                  <input message="tns:GetStudentRequest"/>
                  <output message="tns:GetStudentResponse"/>
            </operation>
      </portType>
```

```
<binding name="Student_Binding" type="tns:Student_PortType">
      <s11:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="getStudent">
            <s11:operation soapAction="getStudent"/>
      <input>
      <s11:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:examples:studentservice"
            use="encoded"/>
      </input>
      <output>
            <s11:body
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:examples:studentservice"
            use="encoded"/>
      </output>
      </operation>
</binding>

<service name="Studnet_Service">
      <documentation>WSDL File for StudentService</documentation>
      <port binding="tns:Student_Binding" name="Student_Port">
            <s11:address
                  location="http://www.sussex.ac.uk/student/">
      </port>
</service>
</definitions>
```

Figure 2.8: An example of Web Service Description Language

The WSDL document includes seven parts including definitions, types, message, portType, binding and service elements, which are described in the following:

- The <type> element is used to define built-in data types that are used by the Web Services. The built-in data types are defined in an XML schema file.

- The <message> element defines the data format of each individual transmission in the communication, in this example, one message represents the "getStudents" request and the other is the response message. Each message can contain one or more parts, which are the parameters of the message.
- The <portType> element is the most important WSDL element. It describes the operations, which are provided by Web Services.
- The <binding> element indicates which communication protocol is used.
- The <Service> element defines the location of the Web Services.

SOAP relies on XML for its message format. In order to pass through the firewall to support interoperable machine-to-machine interaction over the Internet, usually Hyper Text Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP) is adopted as the communication protocol. There are two different layers to secure the SOAP, message level security and transport level security. The transport level security uses layer 3 or layer 4 protocol to protect the data, such as IPSec, SSL, etc. However, it will break the layer 3 security protection if there is an intermediary SOAP node between two end points (Nordotten N. A., 2009). The message level security is working in layer 7, the application layer. Therefore, the message level security is a proper method to protect the message from end to end and the WS-Security 1.1 OASIS standard is adopted to protect SOAP messages.

2.4.3 RESTful Web Services versus SOAP-based Web Services

Fielding (2000) writes that "REST-based architectures communicate primarily through the transfer of representations of resources". This is fundamentally different from the Remote Procedure Call (RPC) approach that encapsulates the notion of invoking a procedure on the remote server. The RPC messages typically contain information about the procedure to be invoked or action to be taken, which is a similar mechanism to a SOAP-based message. The REST-style Web Services or RESTful Web services provider requires a more constrained architectural style, which provides a uniform set of interfaces or operations, including create, retrieve update and delete. However, the

SOAP-based Web Services provide arbitrary or application-specific interfaces, which are invoked by the service consumer. Unlike SOAP-based Web Services, the RESTful Web Services do not provide descriptions of the web service interface, such as Web Services Description Language (WSDL). It must have an out-of-band agreement between a service consumer and a service provider, for instance, sample code and API documentation. Moreover, the RESTful Web Services only support HTTP in the transport layer instead of other transport layer protocols, for example, FTP, SMTP, etc. However, both of the SOAP-based and RESTful Web Services use an XML document to exchange a message between service consumer and service provider. The major characteristics of RESTful Web Services and SOAP-based Web Services are shown in Table 2.6.

Table 2.6: Comparison of RESTful Web Services and SOAP-based Web Services

	RESTful Web Services	SOAP-based Web Services
URI	Resource-based	Method-based
Message Format	XML	XML with SOAP Envelope
Interface Definition	None	WSDL
Method name	Uniform method	Arbitrary method
Transport Protocol	HTTP	HTTP, SMTP, FTP, etc.
Transport Level Security	SSL	SSL
End-to-End Security	None	WS-Security 1.1 OASIS standards

RESTful Web Services is an architectural style for building a distributed application. It is not a standard or a protocol. But the SOAP is a standard, a protocol and it has been

provided with a series of specifications. Both types of Web Services can use HTTP-based authentication plus Secure Sockets Layer or Transport Layer Security (SSL / TLS) to secure the communication between service consumer and service provider. However, SOAP works with extra secure protocols, the WS-* specifications, which support end-to-end message security. But the RESTful Web Service does not have this option and does not provide any relative security standard. Therefore, this research will focus on the security issues of the Web Services, which is implemented using SOAP.

2.5 Web Services Security Challenges

According to the definitions of "44 U.S.C. § 3542", Information security means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide –

A. Integrity, which means guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity;
B. Confidentiality, which means preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information; and
C. Availability, which means ensuring timely and reliable access to and use of information.

Confidentiality, Integrity and Availability (CIA) are three major aspects of security. Security decisions must always be made with an understanding of the threats facing the system to be secured (Zhang, 2010). Singhal A. et al. (2007) identify the elements of security for Web Services as illustrated in Table 2.7. Schwarz J., et al. (2005) also identifies the traditional security threats and security challenges of Web Services as shown in the Table 2.8 and Table 2.9.

The security challenges between service provider and service consumer can be divided into two parts, one is in the communication / transport layer and the other is in the message layer security. Usually, the security issues in the communication layer can be solved by the Secure / Transport Socket Layer (SSL /TLS) if a SOAP message uses HTTP as a communication protocol. Internet Protocol Security (IPSec) or other network layer security protocols can be used to tackle the security issues in the network layer (layer 3). However, this research only focuses on the message layer security of Web Services.

Table 2.7: Element of security for Web Services (Singhal A., et al., 2007)

Element	Description
Identification and Authentication	Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system.
Authorization	The permission to use a computer resource, granted, directly or indirectly, by an application or system owner.
Integrity	The property that data has not been altered in an unauthorized manner while in storage, during processing, or in transit.
Non-repudiation	Assurance that the sender of information is provided with proof of delivery and the recipient is provided with proof of the sender's identity, so neither can later deny having processed the information.
Confidentiality	Preserving authorized restrictions on information access and disclosure,

	including means for protecting personal privacy and proprietary information
Privacy	Restricting access to subscriber or relying party information in accordance with Federal law and organization policy

Table 2.8: Traditional security threats (Schwarz J., et al., 2005)

ID	Security Challenge	Definition
T-01	Message Alteration	The message information is altered by inserting, removing or otherwise modifying information created by the originator of the information and mistaken by the receiver as being the originator's intention. There is not necessarily a one to one correspondence between message information and the message bits due to canonicalization and related transformation mechanisms.
T-02	Confidentiality	Information within the message is viewable by unintended and unauthorized participants. (e.g. a credit card number is obtained).
T-03	Falsified Messages	Fake messages are constructed and sent to a receiver who believes them to have come from a party other than the sender. For example, Alice sends a message to Bob. Mal copies some (or all of) it and uses that in a message sent to Bob who believes this new action was initiated by Alice. This overlaps with T-01. The principle is that there is generally little value to saying a message has not been modified since it was sent unless we know who sent it.

T-04	Man in the Middle	A party poses as the other participant to the real sender and receiver in order to fool both participants (e.g. the attacker is able to downgrade the level of cryptography used to secure the message). The term “Man in the Middle” is applied to a wide variety of attacks that have little in common except for their topology. Potential designs have to be closely examined on a case-by-case basis for susceptibility to anything a third party might do.
T-05	Principal Spoofing	A message is sent which appears to be from another principal (e.g. Alice sends a message which appears as though it is from Bob). This is a variation on T-03.
T-06	Forged claims	A message is sent in which the security claims are forged in an effort to gain access to otherwise unauthorized information (e.g. A security token is used which wasn't really issued by the specified authority). The methods of attack and prevention here are essentially the same as T-01.
T-07	Replay of Message Parts	A message is sent which includes portions of another message in an effort to gain access to otherwise unauthorized information or to cause the receiver to take some action(e.g. a security token from another message is added).Note that this is a variation on T-01. Like “Man in the Middle” this technique can be applied in a wide variety of situations. All designs must be carefully inspected from the perspective of what could an attacker do by replaying messages or parts of messages.
T-08	Replay	A whole message is resent by an attacker
T-09	Denial of Service	Amplifier Attack: attacker does a small amount of work

		and forces system under attack to do a large amount of work. This is an important issue in design and perhaps merits profiling in some cases.

Table 2.9: Security challenges of Web Services (Schwarz J., et al., 2005)

Security Challenge	Definition
Peer Identification	An act or process that presents an identifier to a system so that the system can recognize a system entity and distinguish it from other entities.
Peer Authentication	The corroboration that a peer entity in an association is the one claimed.
Data Origin Identification	An act or process that presents an identifier to a system so that the system can recognize a system entity and distinguish it from other entities.
Data Origin authentication	The corroboration that the source of data received is as claimed.
Data Integrity	The property that data has not been changed, destroyed, or lost in an unauthorized or accidental manner (see [RFC 2828]). It includes transport data integrity and SOAP message integrity.
Data Confidentiality	The property that information is not made available or disclosed to unauthorized individuals, entities, or processes [i.e. to any unauthorized system entity]. It includes transport data confidentiality and SOAP message confidentiality.
Message Uniqueness	The ability to insure that a specific message is not resubmitted for processing.

2.6 Web Services Security Technologies

Security is fundamentally about protecting assets. Assets may be tangible items, such as operations or your customer database (Meier J.D. et al., 2008). Web application or Web Service is one of the assets of a corporation or enterprise. It will expose the internal, private and sensitive information to the public if a service is used or invoked by an unauthorized user. Yamaguchi Y. et al. (2007) identified that when services are composed together, it is important to consider not only the functional requirements but also the nonfunctional requirements. Especially for security, the security requirements for an application are commonly described in a policy like WS-SecurityPolicy. Web Services security is one of the critical areas for research both in industry as well as in academia. In recent years, many security solutions have been invented and implemented including non-standard Web Services security technologies and de facto Web Service security technologies.

2.6.1 De facto Web Services security technologies

OASIS (Organization for the Advancement of Structured Information Standards) is a not-for-profit consortium that drives the development, convergence and adoption of open standards for the global information society. OASIS announced a set of Web Services Security Standards named WS-Security 1.1 OASIS standard, which were approved by the Web Service Security (WSS) technical committee in November 28th 2006. The WS-Security 1.1 OASIS standard consists of one specification and six token profiles, including:

- WS-Security Core Specification 1.1
- Username Token Profile 1.1
- X.509 Token Profile 1.1
- SAML Token Profile 1.1
- Kerberos Token Profile 1.1
- Rights Expression Language (REL) Token Profile 1.1
- SOAP with Attachments (SWA) Profile 1.1.

The WS-Security Core Specification 1.1 utilizes two open W3C- approved standards, XML Encryption and XML Signature to provide message integrity and confidentiality. Because the core specification is designed to be extensible, other existing security mechanisms can be applied and work with the core specification, for instance, the X.509, SAML, Kerberos, Rights Expression Language. Therefore, five security-related token profiles are proposed and approved by the Web Services Security Technical Committee and can improve the security of Web Services.

2.6.1.1 WS-Security Core Specification 1.1

The WS-Security Core Specification is also named "Web Services Security: SOAP Message Security 1.1", which is written by Lawrence K. et al. (2006) and has been published by OASIS. The specification utilizes the W3C-approved XML encryption and XML signature standard to provide SOAP message integrity and confidentially. Moreover, the specification is designed to be extensible. It defined a mechanism for associating different types of security tokens with message content. Therefore, a range of security protocols or user defined security protocols can work with a SOAP message using this specification instead of fixed or limited security protocols. Figure 2.9 shows the conceptual representation of the WS-Security Core specification. The specification is an extension of the SOAP specification for building secure Web services and provides three main features:

1. Define how to send a security token as part of a SOAP message.
2. Provide message integrity feature.
3. Provide message confidentiality.

As shown in Figure 2.9, a security header <wsse:Security> has been added into the SOAP message header <soap:Header> to define a security framework and includes extensibility mechanisms. The security header block defines different tags to contain security-related information for the intended recipient. A recipient will parse the message security information and obtain the details of the processing rules based on the WS-Security core specification.

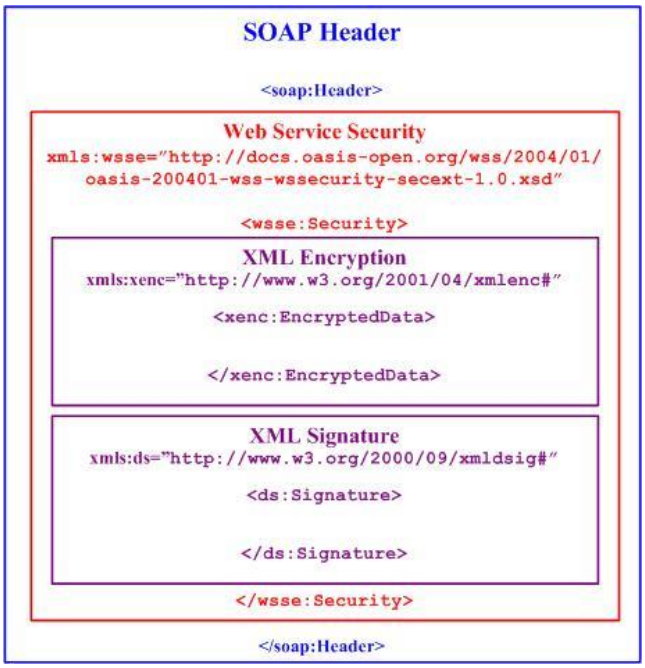

Figure 2.9: The conceptual representation of WS-Security core specification

According to the WS-Security Core Specification, an XML Encryption security element <xenc:EncryptionData>, which is based on the XML encryption standard is added into <wsse:Security> security token block. The encryption element block carries the required information for encryption processing. An example of SOAP message encryption is shown in Figure 2.10. In this example, a public key which is specified in the "<ds:KeyInfo>" element is used to encrypt the data.

```
<S11:Header>

<wsse:Security>

<xenc:EncryptedKey>

        <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>

        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

           <wsse:SecurityTokenReference>

              <ds:X509IssuerSerial>

                     <ds:X509IssuerName>CN=Webster</ds:X509IssuerName>

                     <ds:X509SerialNumber>1325057813</ds:X509SerialNumber>

              </ds:X509IssuerSerial>

           </wsse:SecurityTokenReference>

        </ds:KeyInfo>

        <xenc:CipherData>

           <xenc:CipherValue>

              Gu0KEwiNTGI40UcjTnS6g/4/lsuwhVU5+gaCjOFsihQ4lViD/P7cPkzGBnhEHg
              +oM8SB9AkXyuSnQ0PIMET9gai94kA62Mhm/8f/xaxShNtoGBCZ0Dixc5DGIAMm
              z32wzHC1zbDUDnpQSp6hIP+IOM/SXSQavLZtMaDhxXYTaUtSJxyVHFGNUMfFAm
              WI3H7x26/UIC3zNe0EDUkBPwcWvet5+vMvUnzNFxZyYXsU+yRE+D/+p8qtd3Ax
              xzi5ou+qfQvlIL02d9PqCPt7fvyxqjZtuBN2XheCokUewMVqIG7UmaFfwAsEDG
              thcEIyNGtX8726x3DYxFoKKY1R27f+fw==

           </xenc:CipherValue>

        </xenc:CipherData>

        <xenc:ReferenceList>

           <xenc:DataReference URI="#MsgBody"/>

        </xenc:ReferenceList>

     </xenc:EncryptedKey>

   </wsse:Security>

</S11:Header>
```

Figure 2.10: Example of SOAP message encryption

The <xenc:DataReference> element indicates which message part will be encrypted, the message body is encrypted in this example. A common symmetric key shared by the service provider and the service receiver is used for encrypting the message content. It can also use a one-time symmetric key, which is carried in the <xenc:CipherValue> element of the message. The one-time symmetric key is encrypted by a recipient public key. Therefore, the recipient can decrypt the message using the private key.

The WS-Security Core Specification allows the creation of a digital signature for the SOAP message body. A signature is produced by the message creator and is used to verify message origin and integrity. A message recipient uses the signature to determine whether the message is altered during transmission. An example of a SOAP message signature is shown in Figure 2.11. This specification also allows for multiple signatures and signature formats to be attached to a message. A <ds:Signature> element, which is added to the existing content of the <wsse:Security> header block is used to carry signature-related information. For instance, within the <ds:Signature> element block, the <ds:DigestMethod> element is used to identify the digest algorithm to be applied to the signed object, RSA encryption and SH1 message digest algorithm is used in the example. The encrypted message digest with base64 encoded value is shown within the <ds:DigestValue> element block. The key information, which is used to encrypt the encoded message digest is shown within <ds:KeyInfo> element block.

```
<wsse:Security>

      <ds:Signature  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

            <ds:SignedInfo>

                  <ds:CanonicalizationMethod

                       Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
                       20010315#WithComments" />

                  <ds:SignatureMethod

                        Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />

                  <ds:Reference URI="#MsgBody">

                        <ds:DigestMethod

                              Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />

                        <ds:DigestValue>

                              EVZpDUsThfZQeKXpgijgyLO5PbU=

                        </ds:DigestValue>

                  </ds:Reference>

            </ds:SignedInfo>

            <ds:SignatureValue>

                  RCCAJjax942pxXOlCfazRgeNjVAZT8fRbcIICEyRwAeNljKNB4RvoT0u+g96oD
                  MTtACq5xxcf8cu85cP6+l5yrRizbTDQhhkBfNwkw7VDv/1eqszVkxPd96phNcm
                  TZ8rB2xXaJYgrkWoH0NkBqA5NvCrmH1ETpTtnOudDXTOLBo=

            </ds:SignatureValue>

            <ds:KeyInfo>

                  <ds:KeyValue>

                        <ds:RSAKeyValue>

                              <ds:Modulus>

                                    0Xrzq2zclIckhZQwbHiRqZQf3c4T3YsmWOj7jl9NyUCIrXR
                                    3Hit1Q0kf3zCSzeo56MjEy2b8aw+GqGU+cK2r03NHygfOIB
                                    cjGTioCExlPGuWbKMLIeBMzh5V3o2lsGM89sK8S07egngc4
                                    cRcVN12vnj55i1KbXPeST90E3DoX+k=

                              </ds:Modulus>

                              <ds:Exponent>AQAB</ds:Exponent>

                        </ds:RSAKeyValue>

                  </ds:KeyValue>

            </ds:KeyInfo>

      </ds:Signature>

</wsse:Security>
```

Figure 2.11: Example of SOAP message signature

2.6.1.2 Web Service Security: Username Token Profile 1.1

The Username Token profile is used with the Web Services Security (WSS) specification and describes how to use Username Token. It describes how a web service consumer can supply a username and a password to a service provider for the consumer authentication. A <wsse:UsernameToken> token is used to carry user relative information including username, password, etc.

```
<S11:Header>

     <wsse:Security>

          <wsse:UsernameToken>

               <wsse:Username>John</wsse:Username>

               <wsse:Password Type="http://docs.oasis-
               open.org/wss/2004/01/oasis-200401-wss-username-token-
               profile-1.0#PasswordDigest">

                    NtaIDHV2y+beT9ED5IUUck9dvqE=

               </wsse:Password>

               <wsse:Nonce EncodingType="http://docs.oasis-
               open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
               1.0#Base64Binary">

                    zgniDw==

               </wsse:Nonce>

               <wsu:Created>

                    2012-02-10T16:45:27+0800

               </wsu:Created>

          </wsse:UsernameToken>

     </wsse:Security>

</S11:Header>
```

Figure 2.12: Example of username token use in a SOAP message

An example of Username Token used with the WSS specification is shown in Figure 2.12. It illustrates using a password digest with a nonce and a create timestamp. The digest value of password uses SHA-1 (Secure Hash Algorithm 1) message-digest

algorithm for cryptographic hash function. In order to prevent replay attacks, the following equation is used to produce the password digest.

```
Password_Digest =Base64 (SHA-1 (nonce + created + password))
                                                              (2.1)
```

Where:

- Base64: An encoding scheme that represents binary data in an ASCII string format by translating it into a radix-64 representation.
- SHA-1: Secure Hash Algorithm 1 for message digest.
- Nonce: A random value that the sender creates to include in each Username Token that it sends.
- Created: A timestamp is used to indicate the create time.
- Password: A password that is provided by the service provider.

2.6.1.3 Web Service Security: X.509 Certificate Token Profile 1.1

X.509 is an ITU-T (Telecommunication Standardization Sector of International Telecommunication Union) standard for Public-key and attribute certificate frameworks. The X.509 certificate token profile describes how to use the X.509 authentication framework with the Web Services Security Specification 1.1. The binding information of a Public-key to an entity and a set of attributes are stored in the X.509 certificate as shown in Figure 2.13.

CertificateContent ::= Sequence {

Version	Version Default v1,
serialNumber	Certificate SerialNumber,
signature	AlgorithmIdentifier {{SupportedAlgorithms}},
issuer	Name,

validity Validity,

subject Name,

subjectPublicKeyInfo SubjectPublicKeyInfo,

issuerUniqueIdentifier IMPLICIT UniqueIdentifier OPTIONAL,

subjectUniqueIdentifier IMPLICIT UniqueIdentifier OPTIONAL,

extensions Extensions OPTIONAL

}

Figure 2.13: An Example of X.509 Certificate (ITU-T X.509, 2008)

A <wsse:KeyIdentifier> element that specifies the X.509 subject key identifier of the signing certificate if using references to Subject Key Identifier. A user can be authenticated, the identity of a message sender from a trusted source, named a Certification Authority (CA). A CA is an entity, which issues digital certificates to the ownership of a public key. It allows participants to verify or certify the other user's public key. CA is one of the components in Public-key infrastructure. A public key is bound with respective user identity, which must be unique with each CA domain. A user certificate can be trusted because the user certificate is signed by a well-known CA. Based on the hierarchy of CA, chain of trust, which is an ordered list of certificates is used to validate and verify the certificate issuer. The hierarchy of the CA has a tree structure and the top-most is a root certificate, which is a self-signed certificate. Therefore, a certificate receiver can verify a sender's certificate and all intermediate certificates.

The X.509 Certificate Token Profile describes the syntax and processing rules for the use of the X.509 authentication framework with Web Services Security: SOAP Message Security specification (Nadalin A. et al., 2006c). It describes how to store and carry the sender's X.509 certificate within a SOAP message and which elements are to be used to retrieve key information. The key information can be used for the SOAP message encryption and message signature. According to the X.509 Token Profile, three token

references are supported by WSS: SOAP Message Security including: (Nadalin A. et al., 2006c)

- Reference to a subject Key Identifier - The <wsse:SecurityTokenReference> element contains a <wsse:KeyIdentifier> element that specifies the token data by means of a X.509 SubjectKeyIdentifier reference. A subject key identifier may only be used to reference an X.509v3 certificate."
- Reference to a Binary Security Token - The <wsse:SecurityTokenReference> element contains a wsse:Reference> element that references a local <wsse:BinarySecurityToken> element or a remote data source that contains the token data itself.
- Reference to an Issuer and Serial Number - The <wsse:SecurityTokenReference> element contains a <ds:X509Data> element that contains a <ds:X509IssuerSerial> element that uniquely identifies an end entity certificate by its X.509 Issuer and Serial Number.

An example of a SOAP message, which uses a Binary Security Token to contain the binary X.509 security token data is shown in Figure 2.14. It contains a <wsse:BinarySecurityToken> element and a certificate is presented within this element in binary format. The scope of the signature is defined by a <ds:Reference> element within the <ds:SignedInfo> element.

```
<S11:Header>

      <wsse:Security>

            <wsse:BinarySecurityToken

            EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
            200401-wss-soap-message-security-1.0#Base64Binary"

            ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
            wss-x509-token-profile-1.0#X509" wsu:Id="x509cert00">

               MIIChDCCAe2gAwIBAgIBADANBgkqhkiG9w0BAQUFADAwMQswCQYDVQQGEwJH
               QjEMMAoGA1UEChMDSUJNMRMwEQYDVQQDEwpXaWxsIFlhdGVzMB4XDTA2MDEz
               MTAwMDAwMFoXDTA3MDEzMTIzNTk1OVowMDELMAkGA1UEBhMCR0IxDDAKBgNV
               BAoTA0lCTTETMBEGA1UEAxMKV2lsbCBZYXRlczCBnzANBgkqhkiG9w0BAQEF
               AAOBjQAwgYkCgYEArsRj/n+3RN75+jaxuOMBWSHvZCB0egv8qu2UwLWEeiog
               ePsR6Ku4SuHbBwJtWNr0xBTAAS9lEa70yhVdppxOnJBOCiERg7S0HUdP7a8J
               XPFzA+BqV63JqRgJyxN6msfTAvEMR07LIXmZAte62nwcFrvCKNPCFIJ5mkaJ
               9v1p7jkCAwEAAaOBrTCBqjA/BglghkgBhvhCAQ0EMhMwR2VuZXJhdGVkIGJ5
               IHRoZSBTZWN1cml0eSBTZXJ2ZXIgZm9yIHovT1MgKFJBQ0YpMDgGZQVRFU0B
               VSy5JQk0uQ09ggdJQk0uQ09NhgtXV1cuSUJNLkNPTYcECRRlBjAO

            </wsse:BinarySecurityToken>

            <ds:Signature  xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

                  <ds:SignedInfo>

                        ……

                  </ds:SignedInfo>

                  <ds:SignatureValue>

                        RCCAJjax942pxXOlCfazRgeNjVAZT8fRbcIICEyRwAeNljKNB4Rvo
                        T0u+g96oDMTtACq5xxcf8cu85cP6+l5yrRizbTDQhhkBfNwkw7VDv
                        /1eqszVkxPd96phNcmTZ8rB2xXaJYgrkWoH0NkBqA5NvCrmH1ETpT
                        tnOudDXTOLBo=

                  </ds:SignatureValue>

                  <ds:KeyInfo>

                        <wsse:SecurityTokenReference>

                              <wsse:Reference URI="#x509cert00"

                              ValueType="http://docs.oasis-
                              open.org/wss/2004/01/oasis-200401-wss-x509-
                              token-profile-1.0#X509"/>

                        </wsse:SecurityTokenReference>

                  </ds:KeyInfo>

      </wsse:Security>

</S11:Header>
```

Figure 2.14: Example of SOAP message with an embedded certificate using BinarySecurityToken

2.6.1.4 Web Services Security: SAML Token Profile 1.1

The Security Assertion Markup Language (SAML), developed by the Security Services Technical Committee of OASIS, is an XML-based framework for communicating user

authentication, entitlement, and attribute information. The SAML is an XML-based framework for exchanging security information. This security information is expressed in the form of assertions about subjects, where a subject is an entity that has an identity in some security domain (Maler E., et al., 2003). There are three kinds of statement or functions defined in the SAML, including Authentication, Attribute and Authorization Decision. The authentication, attribute and authorization decision statements can be used to authenticate a subject at a particular time. What attributes are associated with a particular subject. What authorizations are granted or denied to specify subject and to specify resource. The major feature of SAML is to provide a Single Sign-on (SSO) solution, which is used among Web Applications.

The SAML Token Profile defines the use of Security Assertion Markup Language (SAML) assertions as security tokens, which are added into the <wsse:Security> header block. The SAML assertions are used with XML signature to bind the subjects and statements of the assertions to a SOAP message. With the XML signature, the assertion statement integrity can be verified. Moreover, the issuer of the assertion statement can be authenticated. An example of SAML is shown in Figure 2.15 (Monzillo R., 2006). A <wsse:SecurityTokenReference> element is used to indicate, which security token profile is used, the SAML V1.1 is used in this example. <saml:Assertion> element contains assertion statements and can be interpreted as follows:

Assertion **A** was issued at Time **T** by issuer **R** subject to Conditions **C**

The value of the issuer attribute is the unique identifier of the SAML authority. A <saml:NameIdentifier> element refers to a subject and defines the user id format in the “nameidformat” property. In this example, it uses X.509 subject as a Nameidentifier.

```
<S12:Header>
   <wsse:Security xmlns:wsse="...">
      <saml:Assertion xmlns:saml="..."
         AssertionID="_a75adf55-01d7-40cc-929f-dbd8372ebdfc"
         IssueInstant="2003-04-17T00:46:02Z"
         Issuer=”www.opensaml.org”
         MajorVersion="1"
         MinorVersion="1">

         <saml:AuthenticationStatement>
            <saml:Subject>
               <saml:NameIdentifier
                  NameQualifier="www.example.com"
                  Format=“urn:oasis:names:tc:SAML:1.1:nameidformat:
                  X509SubjectName”>
                       uid=joe,ou=people,ou=saml-demo,o=baltimore.com
               </saml:NameIdentifier>
               <saml:SubjectConfirmation>
                  <saml:ConfirmationMethod>
                       urn:oasis:names:tc:SAML:1.0:cm:bearer
                  </saml:ConfirmationMethod>
               </saml:SubjectConfirmation>
            </saml:Subject>
         </saml:AuthenticationStatement>
      </saml:Assertion>
   </wsse:Security>
</S12:Header>
```

Figure 2.15: Example of SOAP message with SAML

2.6.1.5 Web Services Security: Kerberos Token Profile 1.1

Kerberos provides a means of verifying the identities of principles, (e.g., a workstation user or a network server) on an open (unprotected) network (Neuman C. et al., 2005). Kerberos is a computer network authentication protocol and was originally developed for MIT’s Project Athena in the 1980s. It is designed in a client-server model and uses

secret-key cryptography to provide mutual authentication between client and server. Kerberos was developed in 1983, released as open source in 1987 and became an IETF standard in 1993. Nowadays, many operating systems are embedded with Kerberos implementation including Apple Macintosh, Microsoft Windows and UNIX operating systems. The most up to date Kerberos is version 5 and was published in RFC4120 in 2005. Based on the Kerberos authentication process, a client can use the shared secret, which is stored in the Authentication Server (AS) or Key Distribution Center (KDC) to obtain the Ticket Granting Ticket (TGT) and session key from the AS. The TGT and the session key will be used for further communication with the Ticket Granting Server (TGS).

The Kerberos Token Profile describes how to use Kerberos (Kerb) tickets (specifically the AP-REQ packet) with the WSS: SOAP Message Security [WSS] specification, an example is shown in Figure 2.16 (Nadalin A., et al., 2006d). The <wsse:BinarySecurityToken> element with "http://docs.oasis-open.org/wss/oasiswss-kerberos-token-profile-1.1# Kerberosv5_AP_REQ", which is defined in ValueType attribute is used to specify the adoption of Kerberos protocol in SOAP message. A symmetric encryption algorithm is used if the Kerberos ticket is referenced as an encryption key.

```
<S12:Header>
   <wsse:Security xmlns:wsse="...">
      <wsse:BinarySecurityToken EncodingType="http://docs.
         oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
         1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/oasis-
         wss225
         kerberos-token-profile-1.1#Kerberosv5_AP_REQ" wsu:Id="MyToken">
         boIBxDCCAcCgAwIBBaEDAgEOogcD...
      </wsse:BinarySecurityToken>
      ...
      <wsse:SecurityTokenReference>
         <wsse:Reference URI="#MyToken"
            ValueType="http://docs.oasis-open.org/wss/oasis-wss-kerberos-
            token232
            profile-1.1#Kerberosv5_AP_REQ">
         </wsse:Reference>
      </wsse:SecurityTokenReference>
      ...
   </wsse:Security>
</S12:Header>
```

Figure 2.16: Example of SOAP message with Kerberos

2.6.2 Non-standard Web Services security technologies

Although the WS-Security 1.1 OASIS standard has been published to protect a SOAP message between two end-points, there is still significant research into Web Services security. Authentication and authorization are the mechanisms to determine who you are and what you are authorized to do. Both mechanisms are adopted in many information systems and the difference between old and new technology is the implementation. The authentication process may be as simple as a providing a username and password to an authenticating system, or as complicated as using PKI authentication or a Kerberos authentication system. Research reported by Genge B. et al., 2009, proposes a new secure token to extend the WS-security for implementing existing secure protocols such as ISO9798, Kerberos, etc. Other research, such as Damiani E. et al., 2002, designs and

implements an Authorization Filter to provide authentication and authorization features between a client and a SOAP gateway for all the SOAP requests. It also describes new tokens or tags to provide the security features for instance, “userid”, “passwordhash”, “role”, etc. However, it is not a new security mechanism to protect a SOAP message. Authorization is the next process after authentication to determine which web services and methods can be invoked by an authenticating user or a participant. Group-based access control and role-based access control (Sandhu R.S., et al., 1996) are mostly used to control the level of access rights within a system. Other similar models include task-based access (Oh S. and Park S., 2003) and provision-based access control (Kudo M., 2004). These models are static and require a pre-defined access level for each participant.

2.7 Conclusion

In this chapter, it was explained why the Service-Orinted Articture (SOA) is such an important architecture style and an example of an SOA-based system was discussed. The research presented the relationship between SOA and Web Servcices. Two classes of Web Services were discussed in this chapter, which are REST-compliant Web Services and SOAP-based Web Services. REST-compliant Web Services is a resource-based uniform interface Web Service and SOAP-based Web Services is a method-based application-specific interface Web Service. This chapter also covered the major security challenges of Web Services and the currently de facto and non-standards Web services security technologies.

CHAPTER 3

– PARTICIPANT DOMAIN NAME TOKEN PROFILE (PDNT)

3.1 Introduction

The goal of domain names is to provide a mechanism for naming resources in such a way that the names are usable in different hosts, networks, protocol families, internets and administrative organizations (Mockapetris P., 1987). The Domain Name System (DNS) is a hierarchical distributed naming system like a tree structure for network devices, which connect to the Internet or internal network. Each node in the domain naming tree has one or more zone files, which are used to store resource records and is managed by a domain name server. The DNS is implemented in client-server model and maintained by a distributed database system. A DNS client issues a DNS query to a DNS server or DNS resolver using User Datagram Protocol (UDP). The DNS resolver listens on port number 53 to handle the DNS query and returns a Resource Record (RR) to the client. The resource record is the fundamental element in the DNS and each record has its own type. The common format of resource record is specified in RFC 1035 and the types of DNS record are shown in Table 3.1.

Table 3.1: Examples of DNS record types

Type	Type Number	RFC	Description
A	1	RFC 1035	Address Record
LOC	29	RFC 1876	Location Record
MX	15	RFC 1035	Mail Exchange Record
NS	2	RFC 1035	Name Server Record
PRT	12	RFC 1035	Pointer Record
SRV	33	RFC 2782	Service Locator
TXT	16	RFC 1035	Text Record

The domain name is used to uniquely identify Internet resources such as a Web site, and email system, etc. Domain name is human readable and it can be translated to the numerical identifiers, the Internet Protocol (IP) address by the DNS. The IP address is used by a machine and it can be used to locate an Internet service in the worldwide web. For instance, the website of the University of Sussex, http://www.sussex.au.uk is mapped to an IP address of 139.184.32.51. A user inputs an URL to a HTTP user agent,

which is a Web browser usually for browsing a website. A corresponding IP address will be translated by one of the DNS records stored in the DNS server, which is specified or configured in the user computer. A public DNS database is maintained by domain name registrars who are accredited by the Internet Corporation for Assigned Names and Numbers (ICANN). ICANN is a nonprofit organization and was founded in 1998 to oversee Internet-related tasks including Internet Protocol address spaces assignment, top-level domain name space management, etc. ICANN also publishes the complete list of top-level domain registries and domain name registrars. Usually, the local Internet Service Provider (ISP) is the domain name registry and manages the domain name database and the relationship with the domain name applicants. An applicant can also apply for resource records, which are hosted in a local ISP. The rules and regulations of applying domain name and resource records are defined by each local ISP. The valid resource records are forwarded to other DNS systems and are used in worldwide DNS clients.

A Web Services consumer, which is an application or software module sends a service request or invokes a service method using HTTP protocol, an example is shown in Figure 3.1. The HTTP request contains not only the requested information but also the necessary information, which is used for responding back to the requester, for instance, the requester's IP address and port number, "192.168.1.1" and "8888" in this example. Therefore, the requester IP address cannot be faked as the requester wants the result to be sent back. The mapping between the IP address and domain name is stored in the DNS, which is hosted by the Internet Service Provider (ISP). Usually, the ISP is a trusted local telecommunication company. Therefore, the DNS records hosted by the ISP are trusted and reliable. The Participant Domain Token Profile uses one of the DNS resource records, the service resource record to validate the location or domain of participants. The details of the service resource record are described in the next section.

```
Content-Type: application/soap+xml;charset=UTF-8
Content-Length: 44501
Host: 192.168.1.1:8888
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.2 (java 1.5)

<?xml version="1.0" encoding="UTF-8" ?>
<S11:Envelope xmlns:S11="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance" >
   <S11:Body>
      ……
   </S11:Body>
</S11:Envelope>
```

Figure 3.1: An example SOAP message using HTTP

3.2 Service Resource Record

The mapping between IP address and Domain name or other resource records, for instance MX record are stored in the zone file or database of the Domain Name System (DNS). There are many types of record resources stored in the DNS. Each type of resource record is defined Request for Comments (RFC) and has a particular function. For instance, the “A Record”, the type code is “1”, which is used to return a 32-bit IPv4 address by giving a hostname. A Service resource record (SRV RR record) which is defined in RFC 2782 (Gulbrabdsen A., et al., 2000) is one of the DNS resource records and its type code is 33. The SRV RR allows administrators to use several servers for a single domain, to move services from host to host with little fuss, and to designate some hosts as primary servers for a service and others as backups. It is used to define the location of the servers for specified services. For instance, Session Initiation Protocol (SIP) uses the SRV record which is stored in the DNS to find or point to a SIP server, which is listening on TCP port 5060 for SIP services. According to the RFC 2782, the format of the SRV record is:

```
_Service._Proto.Name TTL Class SRV Priority Weight Port Target
                                                        (3.1)
```

Where:

- _Service: is service symbolic name. The name is a unique name, which is legal for SRV lookup in the DNS database.
- _Proto: is the symbolic name of the desired protocol. _TCP and _UDP are the most useful values for this field.
- Name: is the domain this RR refers to.
- TTL: is standard DNS time to live.
- Class: is standard DNS class field.
- SRV: Service record.
- Priority: is the priority of this target host. A client must attempt to contact the target host with the lowest-numbered priority it can reach.
- Weight: is a server selection mechanism. The weight field specifies a relative weight for entries with the same priority. Larger weights SHOULD be given a proportionately higher probability of being selected. The range of this number is between 0 and 65535.
- Port: is the port on this target host of this service. The port number range is between 0 and 65535.
- Target: is the domain name of the target host. There must be one or more address records (A Record) for this name.

The SRV record works with “A Record” because the target field defined in the SRV record is pointing to an address record. Therefore, the corresponding hostname record must be defined in the DNS. The proposed token profile utilizes the SRV record and makes a little change to the meaning of the target field to be a self-defined host instead of the remote host. Although the definition of the target field of RFC 2782 has a minor change, it will not affect the existing usage because of different service names. The proposed token profile can be implemented by using the following SRV record defined in the DNS server.

```
_pdn_tcp.sussex.ac.uk 300 IN SRV 1 1 8080 soap.sussex.ac.uk
                                                    (3.2)
```

Where:

- _pdn : service name for proposed token.
- _tcp : using TCP protocol.
- .sussex.ac.uk : domain of owner.
- 300 : time to live, 300 seconds.
- IN : Internet class.
- SRV : Service record.
- 1 : (0-65535). Lowest is high priority.
- 1 : weight.
- 8080 : port number of target host of desire service.
- soap.sussex.ac.uk : the name of the host that will provide this service. However, it has changed the definition to message sender.

In this example, _pdn is used as a service name and uses TCP (_tcp) as the communication protocol. The domain owner is ".sussex.ac.uk" and "time to live" is 300 seconds, which is the caching time in the DNS. Internet class (IN) and Service record (SRV) must be used in this example. The priority and the weight are both set to 1 to define that it has a high priority and lower weights. The port number is defined in port 8080 and all the requests must come from the corresponding IP address of soap.sussex.ac.uk.

3.3 Classification of WS-Security Token Profiles

There are two layers of security challenges in adopting Web Services: the transport layer security and message layer security. The transport layer security issue is addressed by transport layer protocol. The most widely used transport protocol for Web Services is

HTTP and Hypertext Transfer Protocol Secure (HTTPS), this is one of the mechanisms to satisfy transport layer security requirements. With HTTPS, a user agent or browser adds an encryption layer of SSL/TLS to protect traffic. The objective of HTTPS is to create a secure channel over an insecure channel, which can prevent the eavesdroppers and man-in-middle attacks. However, the primary concern of this research is the message layer security challenge.

The message layer security challenge can be addressed by WS-Security standards if SOAP is adopted to implement a Web Service. The WS-Security standards consist of one core specification and five secure token profiles. The processing model for WS-Security with all five token profiles is no different from other security tokens defined in the WS-Security core specification. The message processor or handler must do the token validation and follow the processing rules, which are defined in related protocol specifications, not in each token profile. All OASIS WSS token profiles involve cryptology, encryption, signature mechanisms or Public Key Infrastructure (PKI) to provide authentication and authorization features. The five secure token profiles focus on two major areas. Who can use the Web Services and what are the permissions. The classification of five WSS token profiles is shown in Table 3.2. Integrity is provided to a SOAP-based Web Service using XML signature. Confidentiality is also provided using XML encryption. Message uniqueness is provided by Username token profile and other security issues are tackled by the other four secure token profiles. However, the location or domain of Web Services participants is not handled and not verified in existing secure token profiles. XML security standards also do not rely on location. Therefore, a new secure token profile is proposed in this thesis to provide additional security for use by Web Services.

Table 3.2: WS-Security token profiles classification

Area	Secure Token Profile
Web Service Authentication (Who)	Username Token Profile X.509 Certificate Profile Kerberos Token Profile Security Assertion Markup Language (SAML) Token Profile
Web Service Authorization (What)	Rights Expression Language Token Profile Security Assertion Markup Language (SAML) Token Profile

3.4 Proposed Participant Domain Name Token

The Participant Domain Name Token Profile (PDNT) is used with WSS: SOAP Message Security specification (WSS). It describes how a participant supplies a Domain Name token as a means of identifying the participant by domain name to authenticate the participant location. In order to use PDNT in a SOAP message, the namespaces which show in Table 3.3 are used and a new element block <pdn:ParticipantDominNameToken> is introduced and added into the <wsse:Security> block. The corresponding schema file of PDNT is shown in Figure 3.2. Within the <pdn:ParticipantDominNameToken> element block, a <pdn:DomainName> element is specified. An example of the use of PDNT is shown in Figure 3.3. It contains a participant or a message sender domain name with the required format and it will be translated to the IP address through the SRV resource record, which is stored in DNS. The details of how to use the proposed token are illustrated in the next section.

Table 3.3: Namespaces are used in PDNT

Prefix	Namespace
S11	http://schemas.xmlsoap.org/soap/envelope/
S12	http://www.w3.org/2003/05/soap-envelope
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd
wsu	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-utility-1.0.xsd
pdn	http://schema.sussex.au.uk/participant-domain-name-token-profile.xsd

```
<xsd:schema targetNamespace="http://schma.sussex.au.uk/participant-domain-name-
token-profile.xsd" xmlns="http://schema.sussex.au.uk/participant-domain-name-token-
profile.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <xsd:complexType name="ParticipantDomainNameTokenType">

      <xsd:annotation>

         <xsd:documentation>

            This type represents a participant domain name token

         </xsd:documentation>

      </xsd:annotation>

      <xsd:complexContent>

         <xsd:sequence>

            <xsd:element name="DomainName"

                  type="xsd:string" minOccurs="1" maxOccurs="1" />

            <xsd:any processContents="lax" minOccurs="0" maxOccurs="unbounded" />

         </xsd:sequence>

      </xsd:complexContent>

   </xsd:complexType>

   <xsd:element name="ParticipantDomainNameToken"

                  type="wsse:ParticipantDomainNameTokenType">

      <xsd:annotation>

         <xsd:documentation>

            This element defines the pdn:ParticipantDomainNameToken element

         </xsd:documentation>

      </xsd:annotation>

   </xsd:element>

  </xsd:schema>
```

Figure 3.2: A schema file of Participant Domain Name Token

```
<soap:Envelope xmlns:soap="..." xmlns:wsse="..."
Xmlns:pdn=="http://schma.sussex.au.uk/participant-domain-name-token-profile.xsd">

   <soap:Header>
      ...
      <wsse:Security>
         <pdn:ParticipantDomainNameToken>
            <pdn:DomainName>
               _pdn_tcp.sussex.ac.uk
            </pdn:DomainName>
         </pdn:ParticipantDomainNameToken>
      </pdn:Security>
      ...
   </soap:Header>
   ...
</soap:Envelope>
```

Figure 3.3: An example SOAP Message using Participant Domain Name Token

In the above example, a service name with protocol type can be found within the <pdn:DomainName> element block, which is _pdn_tcp.sussex.ac.uk. The Web Services participants use the following rule or format to obtain the corresponding SRV resource records. If the result is not saved in local DNS cache before, a DNS client will query a DNS server to obtain the result and keep it in the local DNS cache to improve the performance next time.

```
QNAME = _pdn_tcp.sussex.ac.uk, QCLASS=IN, QTYPE=SRV   (3.3)
```

Based on SRV resource records, a service provider can validate a message sent by a service consumer or vice versa. However, the message receiver must ask the message sender to register the SRV resource records with the local ISP when the Participant Domain Name token is implemented. The high availability can be supported by the SRV resource records if multiple SRV records are defined in the DNS zone file as

shown in Figure 3.4. The value of priority and weight fields are used to provide a combination of load balancing and backup service. The first two records of the example share a priority of 10. Therefore, the weight field is used by a client to determine which server will be used. The value of the target field of the first record is webservice1.sussex.ac.uk. It is pointing to an address record (A record) and is mapped to a corresponding IP address. If more than one IP addresses are mapped to a single address record, round robin DNS feature can be used, which is a load distribution technique.

```
_pdn_tcp.sussex.ac.uk 300 IN SRV 10 10 8080 webservice1.sussex.ac.uk
_pdn_tcp.sussex.ac.uk 300 IN SRV 10 20 8080 webservice2.sussex.ac.uk
_pdn_tcp.sussex.ac.uk 300 IN SRV 20 20 8080 webservice3.sussex.ac.uk
_pdn_tcp.sussex.ac.uk 300 IN SRV 20 30 8080 webservice4.sussex.ac.uk
```

Figure 3.4: Multiple SRV records used to provide load balancing and high availability

3.5 Processing Rules of Participant Domain Name Token

The Participant Domain Name Token (PDNT) works with DNS records, which are stored in the DNS server. Figure 3.5 shows the processing flow of the PDNT as illustrated in the following.

1. A service consumer sends an HTTP request, which contains a SOAP message for the service provider.
2. The service provider receives the HTTP request and parses the SOAP message. A participant domain name can be obtained from the <pdn:ParticipantDomainName> element block. The service provider sends a SRV resource record query with the domain name to a local DNS server.
3. If the corresponding resource record cannot be found in the local ISP, then the local ISP will do the recursive query worldwide. A host name defined in the target field is returned and a second DNS query is executed to lookup the IP

address. A corresponding IP address can be resolved by the local ISP and is returned to the service provider.

4. The location of a service consumer or message sender can be authenticated if the following equation is valid. The results will be returned to the service consumer if PDNT and other secure token profiles are validated.

$$[\text{IP Address}]_{\text{RemoteAddress}} = [\text{IP Address}]_{\text{SA}} \qquad (3.4)$$

Where:

- RemoteAddress can be obtained from a container. The container is a Web Service container or traditional Web server container, in JavaServer Pages (JSP) language, it can obtain the message sender IP address by executing request.getRemoteAddr().
- SA is a two-step procedure. First, it looks up a SRV resource record by giving a standard SRV query string. Second, based on the SRV record, it can obtain a canonical hostname and acquires an IP address by querying an address record in DNS.

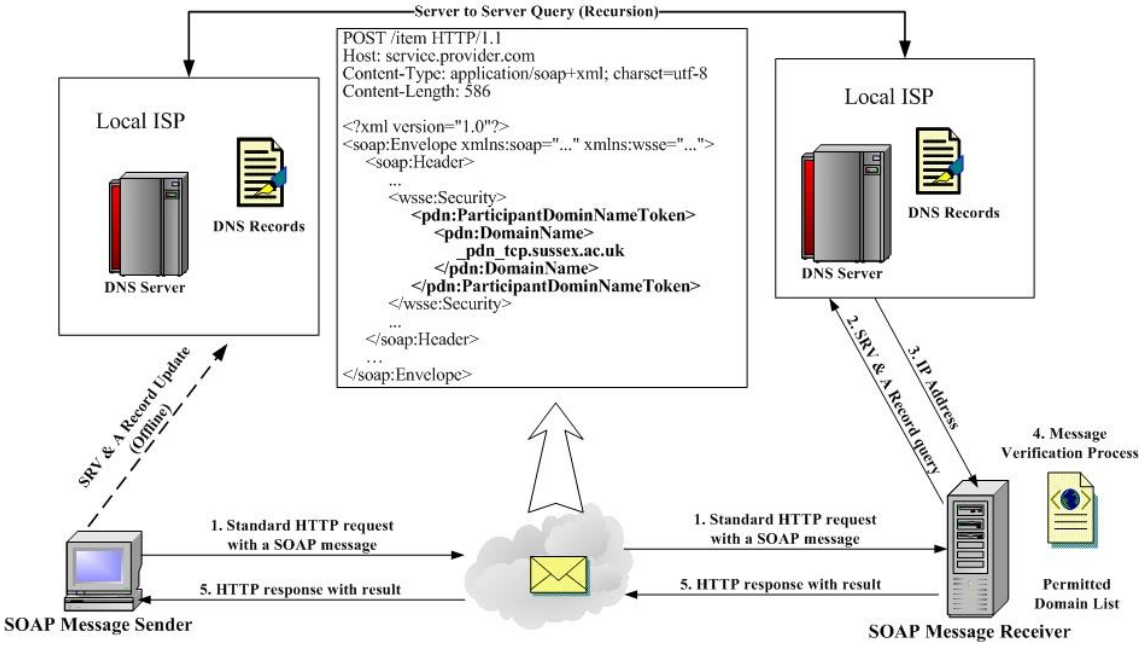

Figure 3.5: The processing flow of the Participant Domain Name Token Profile

To eliminate the overhead for processing an unknown or a fake request, the PDNT is processed before other secure token profiles or other security standards such as X.509 Token Profile, Kerberos Token Profile, etc. In order to support high availability, load balancing and services backup, the DNS server may return more than one SRV resource record. The token processor uses priority and weight fields to determine the precedence for use of the record's data. One of the objectives of PDNT is to reject a fake request as fast as possible, which will reduce the processing resources required to handle illegal requests.

In order to handle a SOAP message which does not use or implement the proposed token, a permitted domain list feature is included in the system. In other words, a permitted domain list is a kind of whitelist, which is used to bypass the processing rule of PDNT. The whitelist can use the IP address or an address record (A Record), which is stored in the DNS server to control who is allowed to use the Web Services. If the IP address is adopted, it will use it to compare the remote IP address directly. Otherwise, the IP resolve step is processed before the comparison. The whitelist will be stored in a secure location such as in a private database or a file in a private folder.

The location-based validation process utilizes one of the existing well-known Internet infrastructures, the Domain Name System. Therefore, it can be trusted and is reliable. In order to parse and verify the proposed token before any other security specifications, arbitrary data encryption cannot be applied to the Participant Domain Name Token profile. It means that the PDNT should be shown in plain text or Base64 encoded. Although the domain name information is sent in plain text, it is not private or sensitive information. The PDNT also must be processed before other secure token profiles because it uses less processing time to validate a message. Depending on the security requirements of the Web Services participants, the PDNT can be used in standalone or with other existing secure token profiles. The performance evaluation of token profiles is illustrated in the next chapter.

3.6 Security Enhancements

Singhal A. et al. (2007) identify that because a Web Service relies on some of the same underlying HTTP and Web-based architecture as common Web applications, it is susceptible to similar threats and vulnerabilities. The fundamentals of security concepts are the Confidentiality, Integrity and Availability, the C-I-A Triad. Based on the research of Web application security issues, Table 3.4 details the foundations of the security element of Web Services and it can be tackled by WS-Security 1.1 OASIS standard as shown in Figure 3.6. Although the basic security elements are solved by secure token profiles, they have a limitation in authentication as illustrated in the next section.

Table 3.4: Fundamental of security element of Web Services

Security Element	Description
Authentication	The identity of a Web Services consumer and provider. Who invokes the Web Services? Who returns the result after invoking a Web Service?
Authorization	The permissions of Web Services operations for a specific Web Services consumer. What can the Web Services consumer do in a Web Service?
Confidentially	The data or information exchange between Web Services consumer and provider remains private and confidential. It cannot be viewed by unauthorized users or eavesdroppers who monitor the flow of traffic across a network. Moreover the intermediary Web Services node cannot also view the information.
Integrity	Web Services Information can be modified or altered by accident or deliberation. Integrity is the guarantee that the data is protected and does not get modified by unauthorized users.

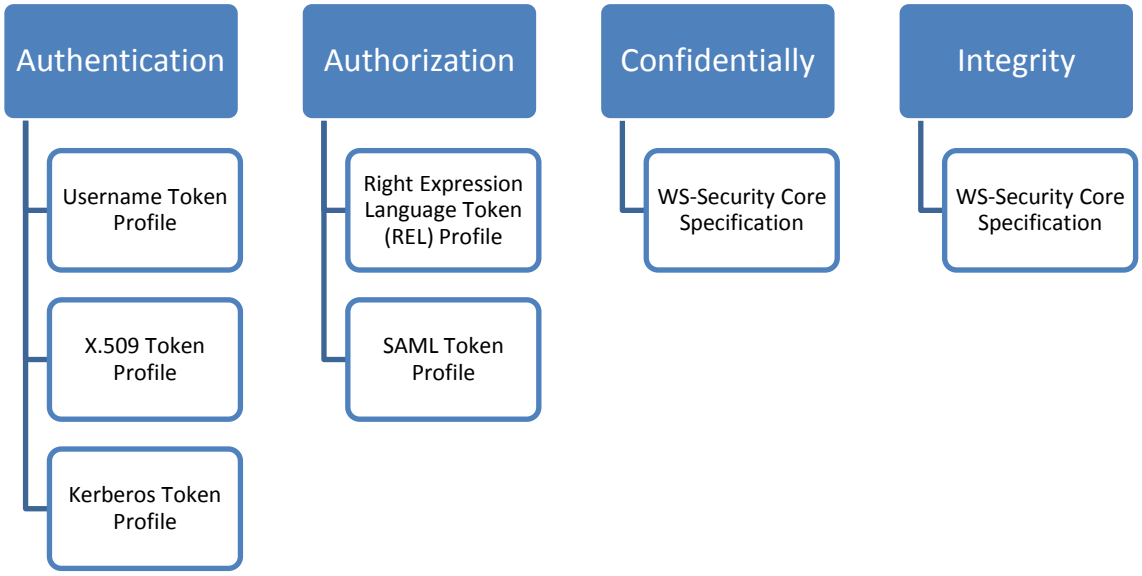

Figure 3.6: The Web application security issues tackled by the corresponding WS-Security token profiles

The Participant Domain Name Token Profile (PDNT) is used to enhance the Web Service authentication security methodology as shown in Figure 3.7. Although the Username token profile, X.509 token profile and Kerberos token profile can also handle the authentication issue; there is a limitation about the authentication of the location of the service consumer and provider. The PDNT can remove this weak point of Web Services authentication. The use of the PDNT has no new message-level threats beyond identified for the PDNT itself. Replay attacks or man-in-the-middle attacks are not threats for the PDNT. One potential threat is the DNS spoofing in ISP DNS. However, this not only applies to PDNT but also to the Web Services architecture. Moreover, the permitted domain list feature of PDNT can prevent an unauthorized domain to invoke Web Services. Other security issues such as message modification and eavesdropping can be addressed by using the integrity and confidentially mechanisms, which are described in other secure token profiles.

A trusted third party is utilized to validate the location of a message sender. This is the Internet Service Provider (ISP), which is the owner of the DNS and is a trusted third party in the processing flow of PDNT. The ISP is an entity or company which

facilitates interactions between two parties who both trust the third party. Most ISPs are telecommunication companies, which is also a trusted third party.

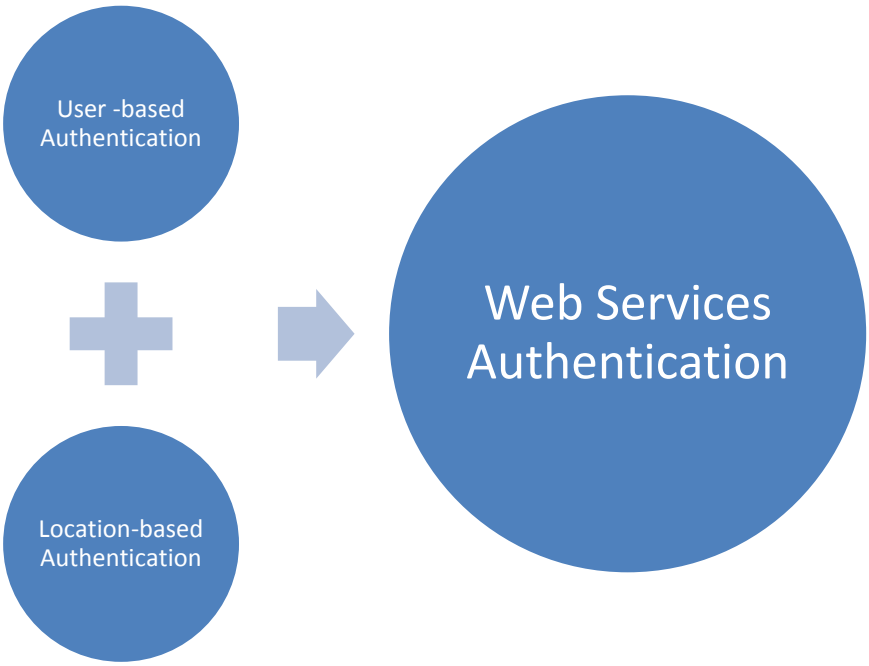

Figure 3.7: The enhancement of Web Services authentication security

3.7 Security Scenarios

There are a total of five secure token profiles for Web Services Security (WSS) Standards after implementing the Participant Domain Token Profile. The following scenarios present the security mechanisms and associated countermeasures that are addressed by the different secure token profiles. The peer authentication security scenario is shown in Table 3.5. It can be tackled using three secure profiles. The message Integrity security scenario is shown in Table 3.6. SOAP message security utilizes XML signature to prevent message alteration. Message confidentiality uses XML encryption to prevent disclosing the content of a message as shown in Table 3.7. The proposed token profile is used to solve location authentication, which is described in Table 3.8.

Table 3.5: Security Scenario 1

Scenario 1	SOAP Message Peer Authentication
Explanation	A Web Service message sender and receiver must be able to authenticate each of other.
Threats	Man-in-middle, Sender Spoofing
Solutions	1. HTTPS X.509 sender authentication 2. Username Token Profile 3. X.509 Certificate Token Profile

Table 3.6: Security Scenario 2

Scenario 2	SOAP Message Integrity
Explanation	A Message Receiver must be able to detect alternation or modification for the content sent from another SOAP node.
Threats	Message alteration, Falsified Messages
Solutions	1. SOAP Message Security – XML Signature 2. Transport layer integrity

Table 3.7: Security Scenario 3

Scenario 3	SOAP Message Confidentiality
Explanation	A Message Receiver must be able to exclusively access confidential content sent from another SOAP node.
Threats	Confidentiality
Solutions	1. SOAP Message Security – XML Encryption 2. Transport layer confidentiality

Table 3.8: Security Scenario 4

Scenario 4	SOAP Message Peer Location Authentication
Explanation	A Web Service message sender and receiver must be able to authenticate the location each of other.
Threats	Location Spoofing

Solutions	1. Participant Domain Name Token Profile

3.8 Security Considerations

There are some security considerations after implementing Participant Domain Name Token (PDNT) profile. It does not mean that the PDNT has security holes or issues. This section discusses the potential attacks and shows the suggestions for reducing security risks.

The PDNT elements defined in a SOAP message are shown in plain text or Base64 encoded. It means that it will expose the sender's domain name. However, the sender's domain name is self-defined information and it is stored in a Domain Name System (DNS) as one of the resource records. Most of the resource records can be queried by a client and it is public information used by Internet users. Therefore, the PDNT uses plain text within an element that does not expose any private or security information to the public.

One of the major roles of PDNT is to use the remote client IP address to compare with an IP address which is retrieved from DNS. However, IP address spoofing may be encountered. IP address spoofing means that the Internet Protocol (IP) packet is created with a forged source IP address. The purpose of IP spoofing is to conceal the identity of the sender or to impersonate another computer system. Usually, routers use the destination IP address to forward packets through the Internet but ignore the source IP address. The source IP address is only used by the destination client when it responds back to the source machine. Forging the source IP address causes the responses to be misdirected to another machine or discarded by intermediate network devices. A SOAP message sending from a forged IP address machine to a Web Service provider will pass the PDNT rules. If the sender modifies the source IP address of all the packets to a permitted IP, it will not expose any sensitive data because the response message cannot send back to a forged IP machine.

3.9 Conclusion

This chapter introduced the PDNT, its theory and its applications. It explains how the PDNT can be used for rejecting an invalid SOAP request, which came from an unknown domain or enterprise.

This chapter classified the defence area of each WS-Security token profile and WS-Security core specification. The research found that the existing Web Service Security standard focuses on two areas, which are user-based authentication and authorization. However, the location of the participant or message sender is not verified. Therefore, a new secure token profile has been designed and proposed, which is the Participant Domain Name Token Profile (PDNT). This chapter presented the PDNT architecture, syntax of the token and the processing rules. The PDNT provides a new security enhancement for SOAP based Web Services, which is location-based authentication.

CHAPTER 4

- PERFORMANCE EVALUATION AND ANALYSIS

4.1 Introduction

Security is one of the successful factors for the use of Web Services on the Internet. Many Web Services security standards are designed and proposed, such as OASIS Web Services Security. It provides end-to-end message security properties including integrity, confidentiality and authentication. However, performance is another vital factor for evaluating a security standard. The processing time is a key indicator for measuring the performance of a new or existing security specification. As more security mechanisms are adopted, additional performance overheads will be added to process the Web Services using CPU processing time, large messages will consume channel bandwidth. In the evaluation, a set of WS-security specifications have been implemented and the Participant Domain Name Token Profile has also been designed and developed. A client-server model is used to evaluate each of the security standards to quantify the effect of the proposed new token.

4.2 Performance Modeling

In order to compare the performance of the Participant Domain Name Token Profile and the five secure token profiles, which are: Usename Token Profile, X.509 Token Profile, SAML Token Profile, Kerberos Token Profile and Rights Expression Language (REL) Token Profile - performance measurements are defined. Juric M.B. et al., (1999) defined a set of performance assessment frameworks for distributed object architecture. The most commonly used performance metrics are response time (R) and throughput (X). This research uses response time or latency time, which is the round-trip time of a message between a sender and a receiver, as a performance metric. The round-trip time does not include a registry lookup such as UDDI or other Web Service retrieval mechanisms. The metric can be used to evaluate the proposed token profile and compare it with other secure token profiles. The latency is defined by the following:

$$\text{Latency} = T_{[\text{Network Layer}]} + T_{[\text{Application Layer}]} \quad (4.1)$$

Where:

- $T_{[Network\ Layer]}$: is the total transfer time of a message between sender and receiver over the network. It consists of many factors. For instance, the number of network devices involved, total delay time in each router or switch, which network protocol is used, the quality of network infrastructure, the distance between sender and receiver, etc.
- $T_{[Application\ Layer]}$: The total processing time spent in the application level, which includes message encoding / decoding, message parsing, security token profiles handling time, total time spent in business logic, database processing time, etc.

However, the total time spent in the network layer is difficult to evaluate over the Internet because it depends on the quality of network transmission and the network devices capacity. Therefore, this thesis only focuses on the time spent in the application layer which is defined as:

$$T_{[Application\ Layer]} = T_{[parsing]} + T_{[security\ token\ profile\ handling]} + T_{[business\ logic]} + T_{[database]} \quad (4.2)$$

Where:

- $T_{[parsing]}$: Total time spent in parsing an input stream into an XML document and converting it to a W3C Document Object Model (DOM), which allows a program to access and manipulate the content of the document.
- $T_{[security\ token\ profile\ handling]}$: Total time spent in handling different secure token profiles and corresponding mechanisms. For instance, XML Encryption, XML Signature, etc.
- $T_{[business\ logic]}$: represents the time spent in business logic.
- $T_{[database]}$: Total time spent in Data Manipulation Language (DML). Most Web Services applications work with a database to query database tables in order to get the result back and return it to requesters.

The proposed token profile is used by a message receiver to authenticate the location of a sender. Other secure token profiles are also used to validate a SOAP message by different mechanisms. In order to compare the performance between proposed token and other existing token profiles, the T[logic] and T[database] factors can be ignored in the performance evaluation. It means that only parsing time and security token profile handling time are taken into account.

Rayns C. et al. (2007) indicates that the CPU consumed by the parsing process is affected by both the overall size of the message, and also by the number of XML elements within the message. Therefore, the latency is not only one evaluation factor but also the additional message size for adopting secure token profiles. For instance, to apply the Username token profile in a Web Service, <wsse:Username>, <wsse:Password>, <wsse:Nonce> and relative elements are used and added into the <wsse:Security> element block in the SOAP message. <xenc:EncyptedKey>, <xenc:EncryptionMethod> and relative elements are used if a SOAP message if it requires encryption. The encrypted data is placed within the <xenc:CipherValue> element block. The SOAP message size depends on which secure toke profiles are adopted. A smaller message size gains a performance advantage, such as the reduction of transfer time and parsing time.

In the performance evaluation, all the measurements are made with identical equipment and environment. In order to eliminate the network and other configuration issues, Web Service consumer and provider are running on an Intel Core2 Duo E8500 computer, using Windows XP Professional with SP3, with 4GB RAM, and restarted before each test case to ensure the same starting conditions. Both the Web Services consumer and Web Services provider are developed using the Java programming language. The Java 2 Platform Enterprise Edition version 1.6.0_21-b07 and SOAP with Attachments API for JAVA (SAAJ) are used for the development and testing environment. A DNS server is installed and running on the same computer. DNS local cache is also enabled to improve the performance of the DNS resolution process, which is used by the

Participant Domain Name Token. All the source codes for the performance evaluation are presented in Appendix B.

4.3 Experimental Results and Analysis

The latency or response time performance metric is used to compare the proposed token profile with three other WS-Security tokens, which include Username token profile, XML Encryption and XML Signature. All test cases are tested with different message sizes. Furthermore, when adopting the proposed token, its performance is compared with three other WSS tokens. A millisecond timescale is used to compute latency for each round-trip time between message sender and receiver.

4.3.1 Evaluation Method and Assumptions

A pure HTTP server has been developed using the Java language and Apache HTTP client JAR is used to develop an HTTP client, which is used to send a SOAP message to the HTTP server. Five java classes have also been developed as shown in the following:

- SOAPEncyrption.java
- SOAPDecryption.java
- SOAPSignature.java
- UserNameToken.java
- PaticipantDomainNameTokenHander.java

All classes are plugged into the pure HTTP server to handle SOAP messages with different types of WSS. RSA 2048-bit is used for the asymmetric key algorithm and AES 128-bit is used for the symmetric key algorithm, SHA is used to create a message digest, which is used in the XML encryption, decryption, signature, and password digest. Latency can be evaluated from the total time spent by the HTTP client when it sends a SOAP request with secure token profiles to an HTTP server, which sends the information back to the HTTP client. In order to evaluate the performance for different sizes of SOAP messages, 100, 200, 300, 400, 500 employee records are included in the message respectively. An example of an employee record is shown in Figure 4.1. Each

SOAP message contains the proposed token and one of the WSS tokens. It means that the message size is identical during the performance comparison between the proposed token and one of the WSS tokens. Each test case is repeated 100 times to obtain the average latency. The average latency time is used for comparison between the proposed token and WSS tokens.

```
<employee>
      <firstName>David</firstName>
      <lastName>Ho</lastName>
      <gender>M</gender>
      <nationality>British</nationality>
      <dateOfBirth>1958-07-01</dateOfBirth>
      <phone>12-543325-8</phone>
      <maritalStatus>Married</maritalStatus>
      <address>No. 7 A Road</address>
      <city>Hong Kong</city>
      <country>China</country>
      <email>david.ho@myService.org</email>
      <empNo>101</empNo>
      <title>Manager</title>
      <department>Accounting</department>
      <hireDate>2005-01-05</hireDate>
</employee>
```

Figure 4.1: An example of an employee record

4.3.2 Test cases design

According to the WS-Security 1.1 OASIS standard, there are five secure token profiles and one core specification. They depend on existing security standards and some of them are using a similar mechanism and algorithm. Therefore, not all the secure token profiles are evaluated. For instance, the X.509 token profile uses a digital signature to verify an X.509 certificate. It is the same as the XML signature token, which is used in

the WS-Security core specification. Therefore, after consolidation, four test cases have been selected, designed and evaluated as shown in the following:

Test Case 1: Proposed Token vs. Username Token

Test Case 2: Proposed Token vs. XML Encryption Token

Test Case 3: Proposed Token vs. XML Signature Token

Test Case 4: Proposed Token vs. XML Encryption with Signature Token

Each test case has been divided into three sub-test cases, which contain (a) the proposed token only; (b) one of the above WS-Security tokens; (c) proposed token with one of the above WS-Security tokens. Each sub-test case is processed a hundred times to acquire the average response time for each sub-test case of each record size.

4.3.3 Message size of each secure token

The size of a SOAP message depends on which secure tokens are used. Different secure token profiles have different element blocks, types, attributes and syntax. For instance, less than 10 elements consist of Username token profile. It is the least number of elements compared to other WS-Security secure token profiles. Therefore, the message size of a SOAP message header using the Username token profile is less than using other WS-Security token profiles. A large SOAP message size will increase the streaming time, network transfer time and parsing time for the message sender and receiver. Table 4.1 and Table 4.2 show the message size and the percentage size increase for each secure token profile. They are the minimum elements requirement for use of each token profile. It means that only compulsory elements are used in each test case and optional elements are not considered.

Table 4.1: Message size of each secure token profile in bytes

Number of Employees Records	100	200	300	400	500
Non-WSS	44,511	88,511	132,511	176,511	220,511
PDNT	44,676	88,676	132,676	176,676	220,676
Username Token	45,054	89,054	133,054	177,054	221,054
PDNT + Username Token	45,204	89,204	133,204	177,204	221,204
Signature Token	45,515	89,515	133,515	177,515	221,515
PDNT + Signature Token	45,665	89,665	133,665	177,665	221,665
Encryption Token	61,946	122,152	182,364	242,576	302,784
PDNT + Encryption Token	62,096	122,302	182,514	242,276	302,934
Encryption + Signature Token	62,935	123,141	183,353	243,565	303,773
PDNT + Encryption + Signature Token	63,085	123,291	183,503	243,715	303,923

Table 4.2: Percentage increase of message size between Non-WSS and each secure token profile

Number of Employees Records	100	200	300	400	500
PDNT	0.37%	0.19%	0.12%	0.09%	0.07%
Username Token	1.22%	0.61%	0.41%	0.31%	0.25%
PDNT + Username Token	1.56%	0.78%	0.52%	0.39%	0.31%
Signature Token	2.26%	1.13%	0.76%	0.57%	0.46%
PDNT + Signature Token	2.59%	1.30%	0.87%	0.65%	0.52%
Encryption Token	39.17%	38.01%	37.62%	37.43%	37.31%
PDNT + Encryption Token	39.51%	38.18%	37.73%	37.51%	37.38%
Encryption + Signature Token	41.39%	39.13%	38.37%	37.99%	37.76%
PDNT + Encryption + Signature Token	41.73%	39.29%	38.48%	38.07%	37.83%

Based on the results in Table 4.2, the message size of the Participant Domain Name Token (PDNT) profile is the minimum and the Encryption Token Profile is the maximum amongst WS-Security secure token profiles. Using WS-Security Token

Profiles, the size of a SOAP message is increased by 1.22% ~ 39.17% when the message contains 100 employee records. However, it only increases by 0.37% if adopting PDNT only. The message size overhead for using PDNT with each WS-Security Token Profile is only increased by 0.34%. It is a major advantage of adopting the PDNT because the message size is much less than adopting other WS-Security token profiles and the overhead adds only a very small increase in message size. With PDNT, the more employee records that are included in a SOAP message, the smaller the relative percentage increase of message size and the overhead is decreased compared to other WS-security token profiles.

4.3.4 Results of Test Case 1

In test case 1, a simple SOAP message, which contains both PDNT and Username token are used for all sub-test cases. Using an identical message header can eliminate the streaming time, network transfer time and parsing time issues because of the different message header size issue. Therefore, identical message size can focus on the processing time of token elements of each sub-test case. The message header of test case 1 is shown in Figure 4.2. Each sub-test case only processes the specified tokens, for instance, the first sub-test only processes the <pdn:ParticipantDominName> element block and other relative elements, other WS-Security token profile elements within <wss:Security> will be ignored.

```
<S11:Header>
   <wsse:Security>
      <pdn:ParticipantDominName>
         <pdn:DomainName>_pdn_tcp.ac.uk</pdn:DomainName>
      </pdn:ParticipantDominName>
      <wsse:UsernameToken>
         <wsse:Username>John</wsse:Username>
         <wsse:Password Type="http://docs.oasis-
         open.org/wss/2004/01/oasis-200401-wss-username-token-profile-
         1.0#PasswordDigest">
            NtaIDHV2y+beT9ED5IUUck9dvqE=
         </wsse:Password>
         <wsse:Nonce EncodingType="http://docs.oasis-
         open.org/wss/2004/01/oasis-200401-wss-soap-message-security-
         1.0#Base64Binary">
            zgniDw==
         </wsse:Nonce>
         <wsu:Created>2012-02-10T16:45:27+0800</wsu:Created>
      </wsse:UsernameToken>
   </wsse:Security>
</S11:Header>
```

Figure 4.2: SOAP message header for test case 1

Table 4.3 lists the results of the comparison between the proposed token and the Username token. A message receiver only uses 7.53 milliseconds to process the Participant Domain Name Token (PDNT), which is less than the message receiver, which uses 8.48 milliseconds to process the Username tokens when the message contains 100 employee records. It shows that PDNT is 12.74% faster to reject a Web Services request if it comes from an invalid location or domain. The third sub test case is to evaluate the performance when both tokens are required to be processed. When a message passes the PDNT validation, it also requires processing of other tokens, the Username tokens in this case. Table 4.3 and Figure 4.3 show that there is a 0.37% processing overhead to process both tokens, this is a very small increase given the additional security provided.

Table 4.3: Latency in milliseconds for test case 1

No. of Employee Records	100	200	300	400	500
PDNT	7.53	14.94	24.09	32.84	41.02
Username Token	8.48	15.93	24.64	33.28	41.05
PDNT + Username Token	8.52	16.23	25.24	34.75	43.34

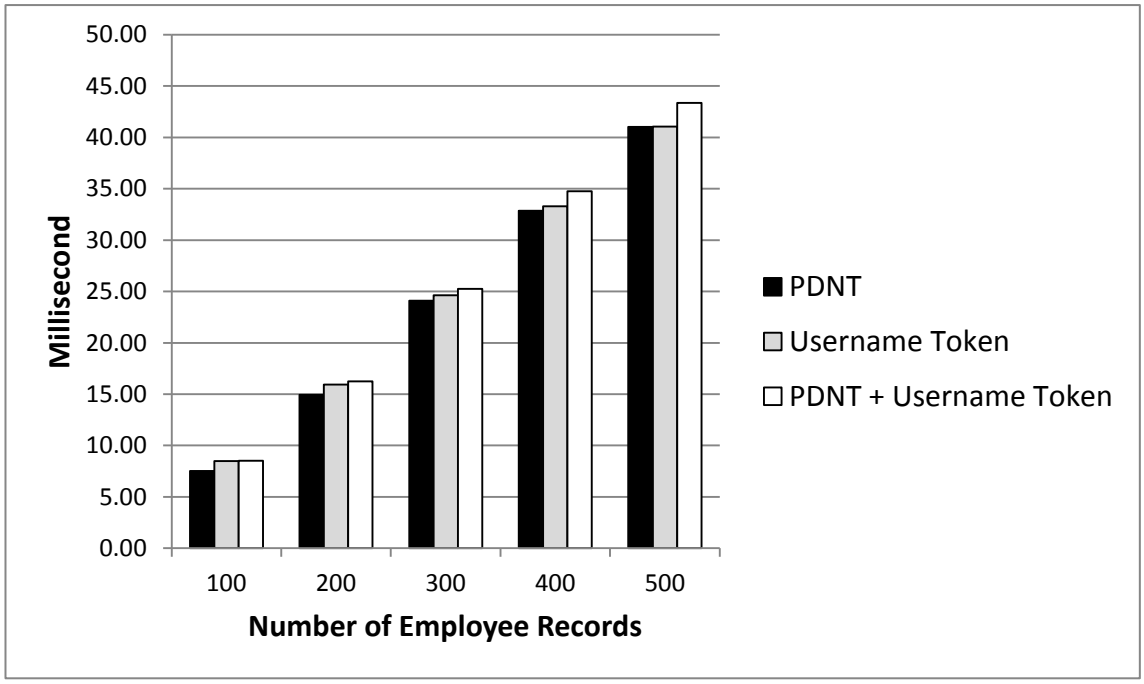

Figure 4.3: Latency in milliseconds for test case 1

4.3.5 Results of Test Case 2

In test case 2, this research assumes that the decryption mechanism defined in the XML encryption standard is used to validate a SOAP message if there are no other authentication methods adopted. The message header of test case 2 is shown in Figure 4.4. As in test case 1, only the secure tokens, which are specified by each sub-test case will be processed. The first sub-test case will handle the PDNT secure tokens. The second sub-test case will process the encryption tokens. The third sub-test case will handle both PDNT and encryption tokens. As shown in Table 4.4 and Figure 4.5,

significant performance gains can be realized by using the proposed token. The processing time of adopting PDNT is more than 10 times faster than the decryption process when a hundred employee records are contained in a SOAP message. A message receiver can reject a message, which is sent from an unauthorized location as fast as the XML decryption processing can be completed. Processing time is increased by only 0.49% to process both tokens when a SOAP message contains 100 employee records. Therefore, the overhead is not significant if both tokens are processed.

```
<S11:Header>
   <wsse:Security>
      <pdn:ParticipantDominName>
         <pdn:DomainName>_pdn_tcp.sussex.ac.uk</pdn:DomainName>
      </pdn:ParticipantDominName>
      <xenc:EncryptedKey>
         <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
         1_5"/>
            <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
               <wsse:SecurityTokenReference>
                  <ds:X509IssuerSerial>
                       <ds:X509IssuerName>CN=Webster</ds:X509IssuerName>
                       <ds:X509SerialNumber>1325057813</ds:X509SerialNumber>
                  </ds:X509IssuerSerial>
               </wsse:SecurityTokenReference>
            </ds:KeyInfo>
            <xenc:CipherData>
               <xenc:CipherValue>
                     DpCR5OuxNJ9dupgjxHoTfO700LbwrtuQ83WVvSUOH4ZATyvKaAutUjW43cD22C
                     QD/qFFU8exrj51hsQQC5NOYKVNuMq1sUCHn26+h0XgmJD8gecz6dClZxA02kjE
                     SCDwYXMMuUCD46mDjGT6Qnf7M2kufaW8vnVJdOZsBF+lQWeGjCVfkmwQatj9UO
                     szLIAfZQ/vti/N4+DUVELqfCamsZz0lvrkvcVFqJVrYV2/i2OMkPxqz8dc7LqF
                     KC+rlXDfl+XVjzXsLmb7knpmhLi6jYO86n91g4Hgo4WMs/jYB6xdHMbHvqY7l2
                     U/xiAk45eV2dwhlxrqib1R0nbLj5SIrg==
               </xenc:CipherValue>
            </xenc:CipherData>
         <xenc:ReferenceList>
            <xenc:DataReference URI="#MsgBody"/>
         </xenc:ReferenceList>
      </xenc:EncryptedKey>
   </wsse:Security>
</S11:Header>
```

Figure 4.4: SOAP message header for test case 2

Table 4.4: Latency in milliseconds for test case 2

No. of Employee Records	100	200	300	400	500
PDNT	4.76	11.04	15.79	21.65	32.99
XML Encryption Token (Decryption Process)	71.00	86.34	103.63	121.78	138.71
PDNT + Encryption Token (Decryption Process)	71.35	87.22	105.51	122.04	139.67

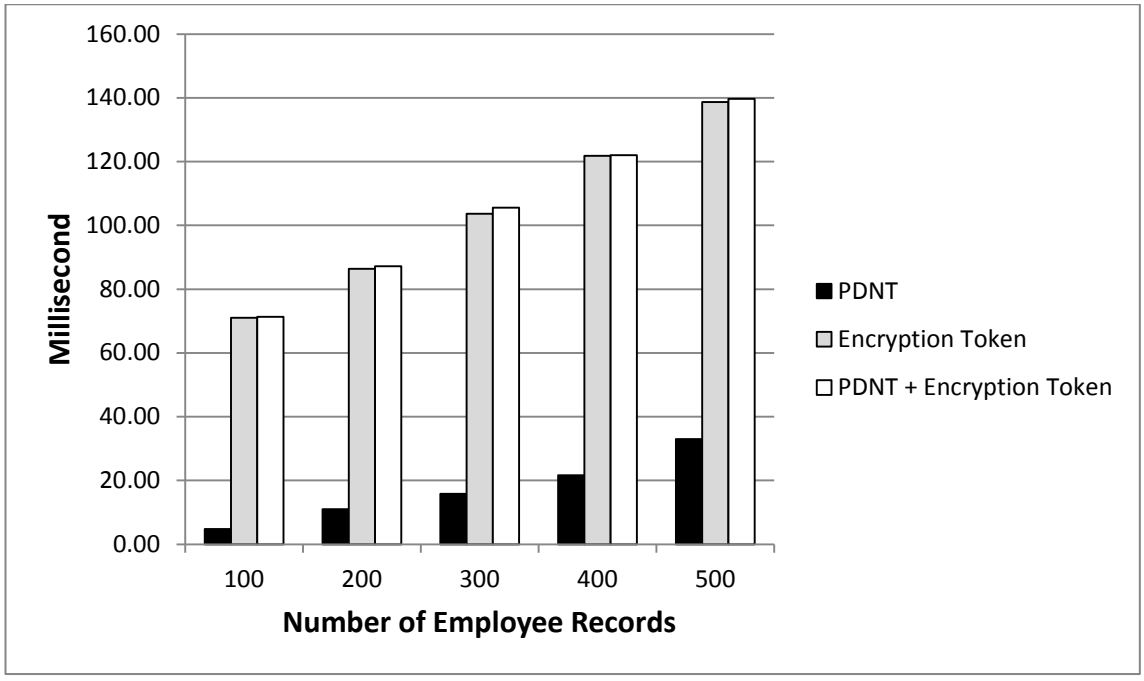

Figure 4.5: Latency in milliseconds for test case 2

4.3.6 Results of Test Case 3

An XML signature is used to ensure SOAP message integrity and the message is sent from a known sender. A SOAP message header for test case 3 is shown in Figure 4.6. Both PDNT and signature tokens are included in the message. Three sub-test cases will be evaluated including the processing time of the PDNT, signature token and PDNT with signature token. Table 4.5 and Figure 4.7 show the performance results of adopting the PDNT, signature token, and PDNT with signature tokens. According to the results

of test case 3, it is 37.71% ~ 62.35% faster on average if adopting PDNT compared to the signature token. The overhead of adopting both tokens is 0.40% ~ 0.43% on average. Therefore, adopting the proposed token has a performance advantage as it can refuse an invalid SOAP message faster.

```
<S11:Header>

   <wsse:Security>

      <pdn:ParticipantDominName>

         <pdn:DomainName>_pdn_tcp.sussex.ac.uk</pdn:DomainName>

      </pdn:ParticipantDominName>

      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

         <ds:SignedInfo>

            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-
            xml-c14n-20010315#WithComments"/>

            <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
            sha1"/>

            <ds:Reference URI="#MsgBody">

               <ds:DigestMethod
               Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

               <ds:DigestValue>yU/ypTa0GSC0kjjf23jgkSxxYf0=</ds:DigestValue>

            </ds:Reference>

         </ds:SignedInfo>

         <ds:SignatureValue>

            IlhHY/0hHFjaRbEMNpBL6urRDUhhboVv9XGb3AcYYLJyel0bNvpdPfBZNLMq7HyNtxEmrfJ
            fghj1IWYAmrsOB7J9bZFSa1UkZpaCgpnhyC4e5dlCtZTK/CJTSkfuQ1RXSQrxHjfh1o9+SO
            fpToTFMMaHRnY+/lAMRCOwhj725Xo=

         </ds:SignatureValue>

         <ds:KeyInfo>

            <ds:KeyValue>

               <ds:RSAKeyValue>

                       <ds:Modulus>

                          miAhwTQHAyCYts1dP4WdEQFvg83cUFoiymuPPXksyYcU+/1X03Z1ac7A4d
                          P/4U6+IRL2+J8cRvMbdy+X1kI/vEYDw1yI4T4snB7XOGxbDi0D80FYUD6a
                          /99e3nr098aT4sVF8eJfD6KUYzFiMP48CAmS59UaSv6AiK/9Le1/7I0=

                       </ds:Modulus>

                  <ds:Exponent>AQAB</ds:Exponent>

               </ds:RSAKeyValue>

            </ds:KeyValue>

         </ds:KeyInfo>

      </ds:Signature>

   </wsse:Security>

</S11:Header>
```

Figure 4.6: SOAP message header for test case 3

Table 4.5: Latency in milliseconds for test case 3

No. of Employee Records	100	200	300	400	500
PDNT	8.22	15.10	24.00	28.69	32.92
XML Signature Token	11.32	20.55	31.61	45.08	53.45
PDNT + Signature Token	11.37	20.80	31.80	45.49	53.67

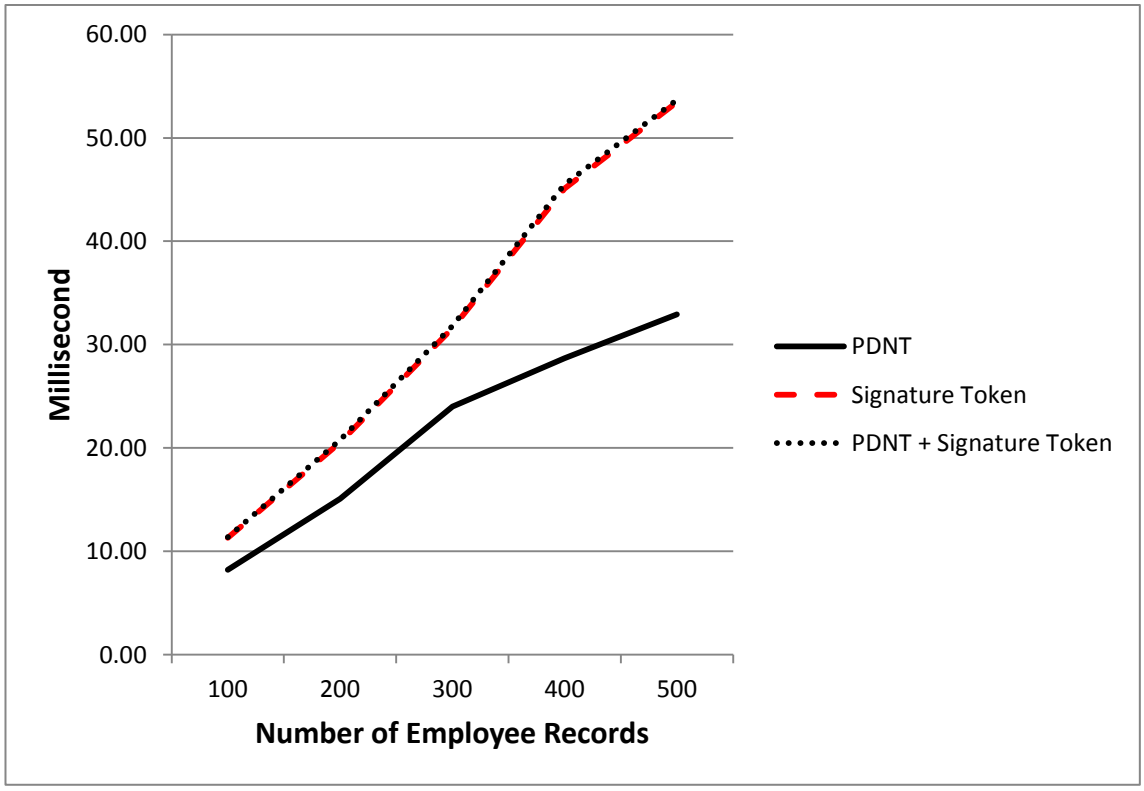

Figure 4.7: Latency in milliseconds for test case 3

4.3.7 Results of Test Case 4

Some Web Services applications require more security mechanisms to protect the information between sender and receiver. Therefore, more than one secure token profile is adopted in a SOAP message. Figure 4.8 shows the message header of test case 4. In test case 4, PDNT, XML encryption and XML signature tokens are used in a single SOAP message. The first sub-test case will only handle the PDNT token. The second sub-test case processes both encryption and signature tokens. The last sub-test case

processes all three secure tokens, the PDNT, encryption and signature tokens. The result of test case 4 is shown in Table 4.6 and Figure 4.9. It shows that the PDNT is faster from 356.30% to 1417.62%, on average if only adopting PDNT compared to the encryption with signature token. The overhead of adopting all three tokens is from 0.11% to 0.78% on average. Therefore, less than 1% overhead can gain an additional security feature if using the PDNT.

```
<S11:Header>

   <wsse:Security>

      <pdn:ParticipantDominName>

         <pdn:DomainName>_pdn_tcp.sussex.ac.uk</pdn:DomainName>

      </pdn:ParticipantDominName>

      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

         <ds:SignedInfo>

            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-
            xml-c14n-20010315#WithComments"/>

            <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
            sha1"/>

            <ds:Reference URI="#MsgBody">

               <ds:DigestMethod
               Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

               <ds:DigestValue>yU/ypTa0GSC0kjjf23jgkSxxYf0=</ds:DigestValue>

            </ds:Reference>

         </ds:SignedInfo>

         <ds:SignatureValue>

               XQ85p6k/Q8xAdA6+bLUiGeSGe4DPe7WGAgfRmdD6QDpGIDIUS/z+4Fn2jSCxeiCq1wJn
               Jjii/R8Hdhe2bWFQNQ7qlPoniuRTHL0TzTUkFBRQw7o+CHvATPokNWs0JmlheiMxdjKI
               kfXts6wLLN5s4faJDmQXFK1bkPETQt1CQ7o=

         </ds:SignatureValue>

         <ds:KeyInfo>

            <ds:KeyValue>

               <ds:RSAKeyValue>

                  <ds:Modulus>

                       xsZaGU62rvUPpYcCx4B2c0Z03K3CNQnEgfM7jA7zngvD3GkmkqeqR587vp7pD
                       F62migJ0+D2letCfyq5H3RYkNVukaVkbpSSQFzesJu+WyCg227O2LeudT58ie
                       v/7H24wJxcfr5oe3PltL0/rnHZ4nz/c5jXH4HmH/CpzBqd1f0=

                  </ds:Modulus>

                  <ds:Exponent>AQAB</ds:Exponent>

               </ds:RSAKeyValue>

            </ds:KeyValue>

         </ds:KeyInfo>

      </ds:Signature>
```

```
      <xenc:EncryptedKey>

         <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-
         1_5"/>

         <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

            <wsse:SecurityTokenReference>

               <ds:X509IssuerSerial>

                  <ds:X509IssuerName>CN=Webster</ds:X509IssuerName>

                  <ds:X509SerialNumber>1325057813</ds:X509SerialNumber>

               </ds:X509IssuerSerial>

            </wsse:SecurityTokenReference>

         </ds:KeyInfo>

         <xenc:CipherData>

            <xenc:CipherValue>

                  JvKOLNz6qa52Wt2eq/9ECBIb2sjndjm1UzaY9wWBh8jdRPd0zn3rNk3DTNKxpqn+K
                  Q0qf5mmKMx+SLI7qyE+19Qa7bvsfGxaGnOTGIoYnxduEg02ypHFGtt1pHF3g9L+zu
                  20TBXj7aPB5EowCsUz1xjRI9PKKdpjGkCqQvLUeTalCOSAUp0hcpYqLklVZdnipOC
                  rng+CVuuoJ63/7SniC1RmzzWlX4LBBYucY4gPWiE/WURlb92sz1oHuTgr2BmFGDzY
                  RqOZcid80NV6OE1o5a5zcwsdibVgjjZogO5Wedv+vXjoDY+4gGAsCnlLWSzTkTof/
                  ARMzIjfRidyrrxjRA==

            </xenc:CipherValue>

         </xenc:CipherData>

         <xenc:ReferenceList>

            <xenc:DataReference URI="#MsgBody"/>

         </xenc:ReferenceList>

      </xenc:EncryptedKey>

   </wsse:Security>

</S11:Header>

</S11:Header>
```

Figure 4.8: SOAP message header for test case 4

Table 4.6: Latency in milliseconds for test case 4

No. of Employee Records	100	200	300	400	500
PDNT	4.88	9.14	15.33	22.33	32.11
XML Encryption + Signature Token	74.10	92.16	113.62	128.25	146.50
PDNT + Encryption + Signature Token	74.56	92.50	114.51	130.84	146.66

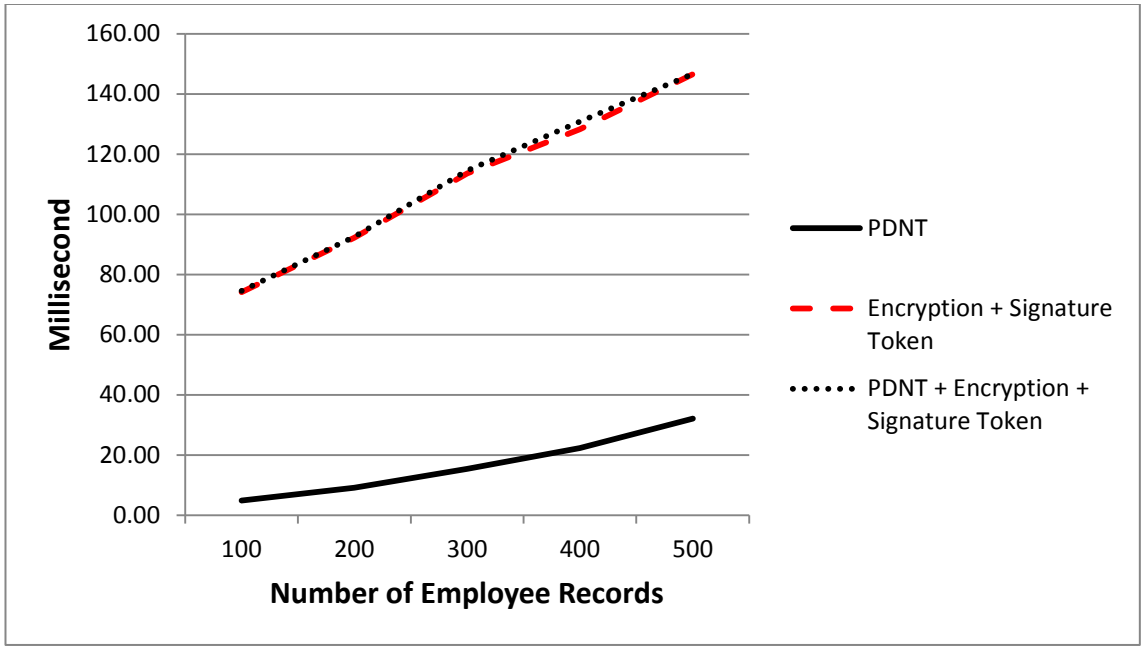

Figure 4.9: Latency in milliseconds for test case 4

4.3.8 Results Analysis

The latency of each test case includes the time for the message transfer between the two end-points, XML parsing; secure token parsing; secure token processing and DNS lookup for PDNT. However, the DNS lookup is not a time consuming process because it always uses local server DNS cache after first time querying. The message transfer time between client and server is also not a major consideration in this testing environment because the client and server process is running on the same computer. According to the results of four test cases, all results, which only adopt the PDNT token are significantly faster than other secure token profiles. Moreover, the overhead of adopting PDNT tokens with other secure token profiles is minor. Therefore, the PDNT has a performance advantage.

4.4 Advantages of Participant Domain Name Token

Three advantages accrue from the proposed Participant Domain Name Token (PDNT). First, it provides one more security feature, which the WSS token profiles do not include. The PDNT works with DNS to provide location or domain validation of a SOAP message. Second, the PDNT has a performance advantage when compared with WS-Security token profiles. The overheads of XML signature, XML encryption and the five secure token profiles are significant. Therefore, using the proposed token, a service provider can reject an unknown domain or faked SOAP request as soon as possible. The proposed token is parsed and processed before the WS-Security core specification and other secure token profiles are processed. Therefore, it can save server resources such as CPU time, memory, energy, etc and handle more valid SOAP requests. Based on the evaluation of the four test cases, it generates less than 1% overhead to adopt PDNT, this is not a significant cost. Third, a permitted domain list feature is also provided. It acts like a white list or approved list to control "Where" can invoke the Web Services method. The PDNT is simple and easy to implement compared with other Web Service security specifications and token profiles.

4.5 Conclusion

This chapter introduced a performance model for measuring the PDNT and WS-Security token profiles because the performance is another vital factor to increase the spread of a new security standard. Four test cases were seleted and designed in this chapter. Each test case has been devided into three sub-test cases to give a detailed comparsion. The total parsing time and the total token handling time are adopted as performance metrics. According to the experimental results of the four test cases, the research found that the PDNT is significantly faster than WS-Security token profiles. Moreover, the processing overhead for adopting the PDNT with the WS-Security token profiles is minor. Therefore, the PDNT gains significant performance advantages.

CHAPTER 5

– DESIGN AND IMPLEMENTATION OF PARTICIPANT DOMAIN NAME TOKEN

5.1 Introduction

The Web Services protocol stack is a set of protocols which are used to define; discover and implement the Web Services as illustrated in Figure 5.1. The core protocol stack consists of five layers. The first layer is the transport layer, which is responsible for transporting messages between applications, the service consumer and service provider. The second layer is responsible for encoding a request or responding to messages in a common XML format. The third layer is responsible for describing the public interface to specific Web Services; centralizing services into a common registry and providing publishing and finds functionality, for instance, what methods can be invoked. It also encodes a request and request message to SOAP format. The fourth layer is responsible for the enhancement of the SOAP message, including the message security, message reliability, etc. The top layer is responsible for the services orchestration. It uses Business Process Execution Language (BPEL), which is an orchestration language for the Web Services interactions. Using the BPEL, the business processes can be defined in XML-based language and exchange messages not only for the internal system but also with the outside system.

To implement a Web Service, many technologies are involved. XML is a core technology to encode the information. Web Services are a mix of XML and HTTP protocol that can convert a message into a Web application. It uses XML to encode methods, parameters and other information exchanges between service consumer and service provider. SOAP, UDDI and WSDL are XML-based protocols and independent of the OS platform and programming languages. The metadata of Web Services is defined by the WSDL and listed and discovered by UDDI. WS-* standards are the enhancements of Web Services including security, reliability, policy, atomic transaction, coordination, etc.

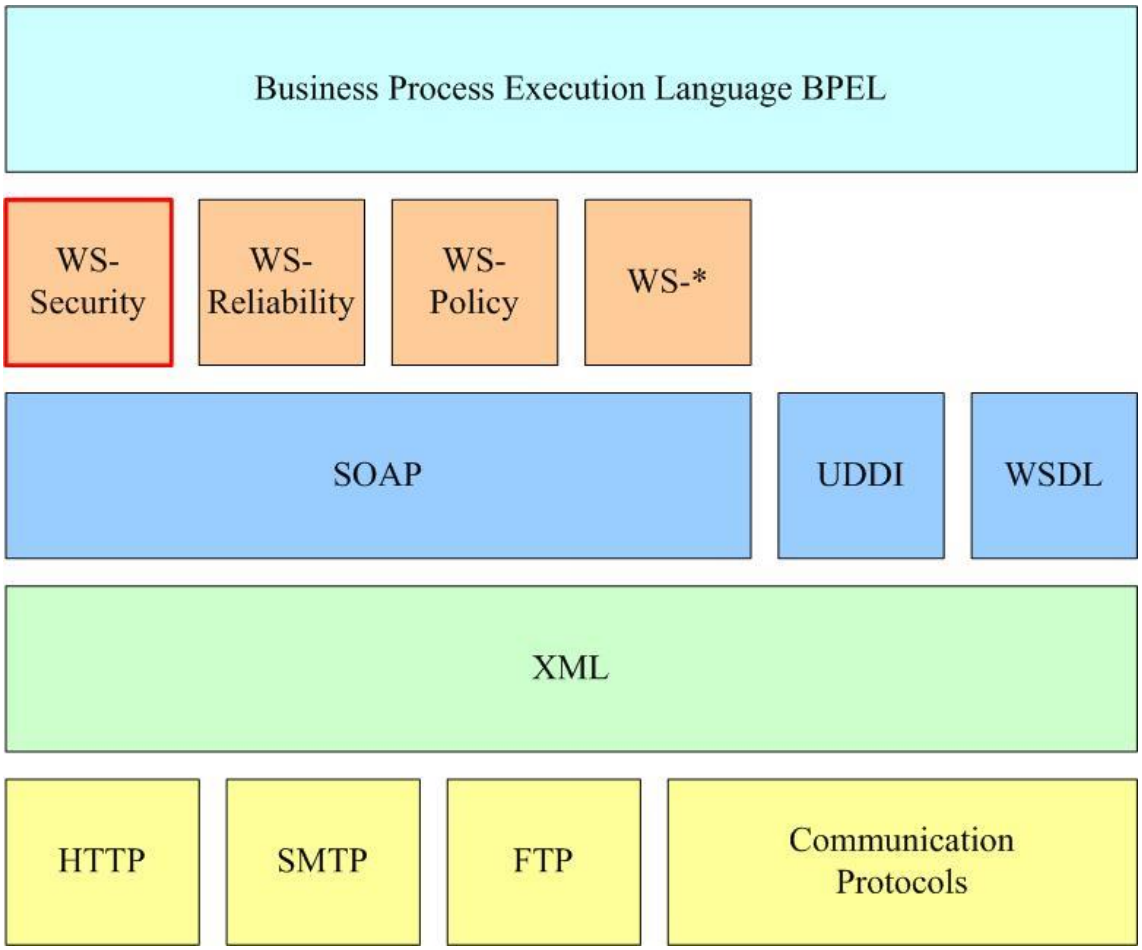

Figure 5.1: The Web Services protocol stack

5.2 Building Web Services Platform

There are two major platforms or frameworks to implement the Web Services, one is J2EE (Java Platform, Enterprise Edition) and the other is Microsoft .NET. J2EE provides APIs and runtime environment for developing and running enterprise software, including network, Web Services, etc. J2EE extends the J2SE (Java Platform Standard Edition), which provides APIs for interacting with file systems, network, graphic interfaces, etc. The .NET Framework is a software framework developed by Microsoft which runs on Microsoft Windows. It provides many libraries for developers to handle the user interface, database access, cryptography, Web Services, etc. Although both platforms support Web Services, the J2EE is platform gives independence. Therefore, J2EE is adopted as the development platform for this research.

5.2.1 Web Container

A Web Server, which is used to host static Web pages is replaced by a Web container or modules-based Web server. Microsoft Internet Information Server (IIS) and Apache HTTP server are the most commonly used Web servers. In "Java World", a Web container or Servlet container is an application running on the Web server to handle or process Java Servlet and Java Server Pages. The container is an independent environment and designed to run Java coding on a Web server. Apache Tomcat is one of the most popular Non-commercial Web containers and Oracle WebLogic is a commercial Web container. Modules based Web server is another solution to add website functionality such as: Microsoft Internet Information Server (IIS). IIS has many modules to form an application pool to provide different functionalities. Therefore, to support Web Services, a Web container or modules based Web server must be used. Web container or Servlet container is used because Java is the programming language used in this research. Apache Tomcat is also used for the development and testing environment. It is an open source software implementation of the Java Servlet and JavaServer Pages technologies. Tomcat is an application server and runs behind the HTTP server. It means that all the requests are handled by the HTTP server and passed to Tomcat when they come across the intended Servlet container.

5.2.2 Java Web Services API

Java API for XML Web Services (JAX-WS) is a technology for building Web Services as shown in Figure 5.2. JAX-WS allows developers to write message-oriented as well as RPC-oriented Web Services. It uses XML-based protocol, SOAP is used to build a Web service, it hides the complexity of the SOAP for the application developer. With JAX-WS, an application developer does not need to parse and form a SOAP message. However, in order to add a new security token profile on top of SOAP message, the SOAP with Attachments API for Java (SAAJ) is used. The SAAJ goes on behind the scenes in JAX-WS handlers and JAXR implementations. The advantages of using SAAJ are shown in the following:

- Application developer can use the SAAJ API to write a SOAP message directly

- The SAAJ API allows a developer to make a method call by doing XML messaging.
- Send and receive a SOAP message over the Internet using SAAJ.
- It conforms to the SOAP 1.1 and SOAP 1.2 specifications.

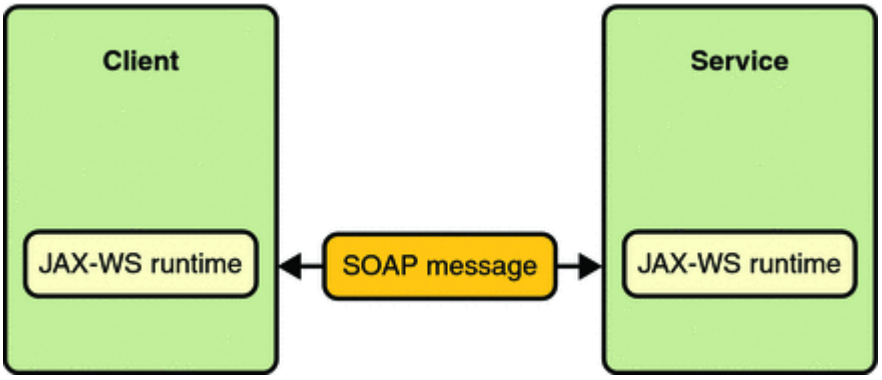

Figure 5.2: Communication between a JAX-WS Web Service and a Client (Oracle, 2010)

The SAAJ 1.3 specification is implemented in the javax.xml.soap package and provides all the necessary APIs for creating, populating and sending request-response SOAP messages. Figure 5.3 illustrates the structure of a SOAP message with no attachments and an example for creating a SOAP message using SAAJ is shown in Figure 5.4. The SAAJ is based on the SOAP standards and provides APIs for the developer to create an XML message that conforms to the SOAP 1.1 or 1.2 specifications.

SAAJ is an API-based SOAP toolkit and can be used to create, read or modify SOAP messages. The APIs include classes and interfaces that parse and handle SOAP elements including SOAP envelope, header, body, and fault. SOAPMessage class is a root class SOAP message. Based on the SOAP specification, there are three major elements within a SOAP message including envelope element, header element and body element. SOAPEnvelope, SOAPHeader and SOAPBody are the three interfaces defined in SAAJ to represent the corresponding element.

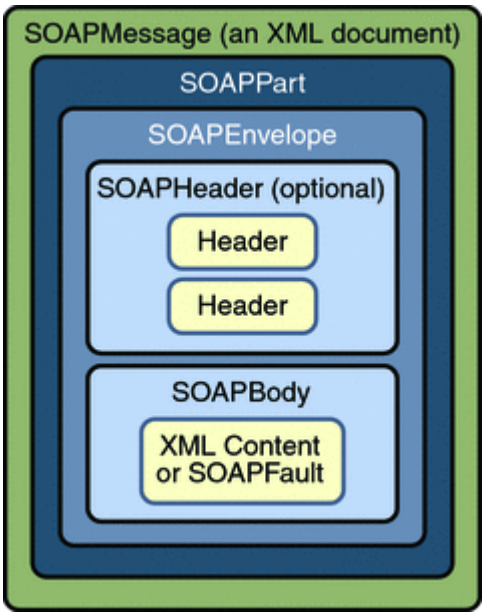

Figure 5.3: SOAPMessage object with no attachments (Oracle, 2010)

```
public static SOAPMessage getSOAPMessage(InputStream in)
   try{
      MessageFactory factory = MessageFactory.newInstance();
      MimeHeaders mimeHeaders = new MimeHeaders();
      mimeHeaders.addHeader("Content-Type","text/xml; charset=UTF-8");
      SOAPMessage soapMessage=factory.createMessage(mimeHeaders, in);
      soapPart = soapMessage.getSOAPPart();
      soapEnvelope = soapPart.getEnvelope();
      soapHeader = soapEnvelope.getHeader();
      soapBody=soapEnvelope.getBody();
   }catch(Exception e){
      e.printStackTrace();
}
```

Figure 5.4: An example of creating a SOAP message using SOAP with Attachments API for Java (SAAJ)

5.3 Web Services Implementation Approaches

There are two approaches that are used to implement a Web Service from scratch as illustrated in Figure 5.5. The first approach is termed “bottom-up” and the other is

termed “top-down”. In the bottom-up approach the source codes or Java methods are implemented before writing the service description, the Web Services Description Language (WSDL). The “top-down” approach starts with writing a contract or WSDL and implements all necessary Java methods based on the Web Services description.

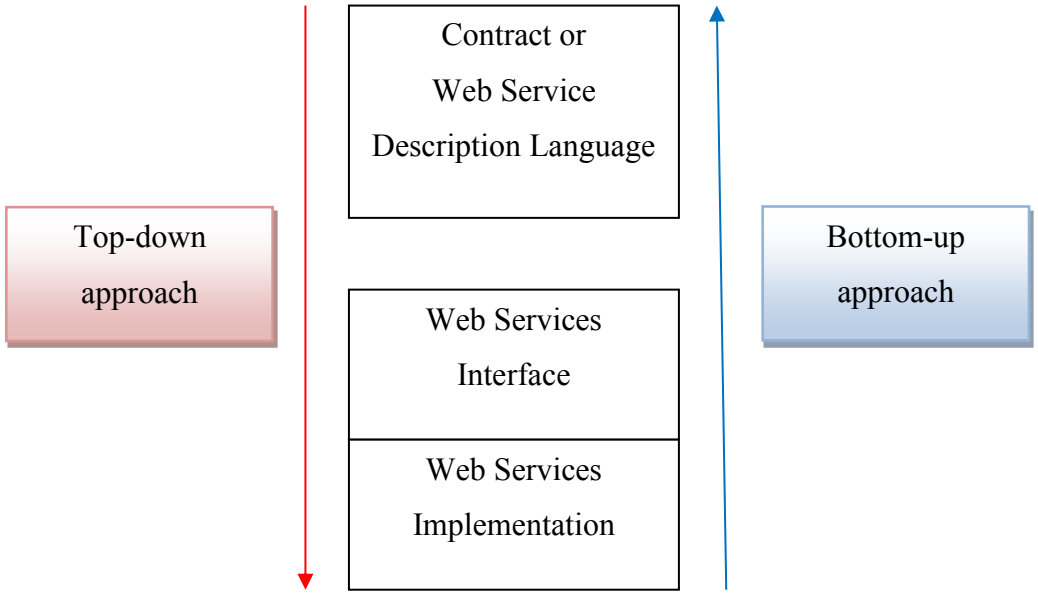

Figure 5.5: Top-down and button-up implementation approaches to Web Services

The bottom-up approach provides a quick and easy way to implement a Web Service from existing coding. However when the Web Service is implemented from a developers point of view, it may not address the needs of the consumers. The top-down approach can address the consumer needs from the outset. It requires that the service provider discusses the service contract with the consumer before any application coding begins. It is designed from a business point of view. However, it may have interoperability issues, for instance: data types, encoding and binding issues between systems. The bottom-up strategy avoids the extra cost, effort, and time required to deliver services via a top-down approach, it ends up imposing an increased governance burden as bottom-up delivered services tend to have shorter lifespans and require more frequent maintenance, refactoring, and versioning. The top-down strategy demands more of an initial investment because it introduces an up-front analysis stage focused on the creation of the service inventory blueprint. Service candidates are individually defined as part of this blueprint so as to ensure that subsequent service designs will be highly normalized, standardized, and aligned Erl T (2005). Therefore, there are no best

approaches to implement a Web Service. It depends on the enterprise architecture, existing systems architecture and the business requirements.

5.4 Implementation of Web Services in Bottom-up Approach

Eclipse IDE, which is a software development environment, Java, Axis2 plug-in and Tomcat are used to implement an example of a Web Service using the Bottom-up approach. A student Web Service is designed and developed for this demonstration as shown in Figure 5.6 and Figure 5.7.

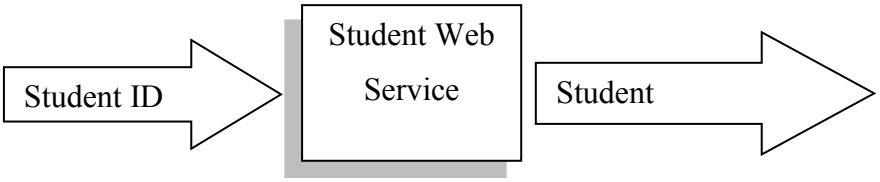

Figure 5.6: Student Web Service

```
package uk.ac.sussex.webservices.method;
import uk.ac.sussex.dao.StudentDao;
import uk.ac.sussex.entity.Student;

public class StudentService {
   public Student getStudnet(long studentId){
      StudentDao studentDao=new StudentDao();
      Student student=studentDao.getStudentById(studentId);
      return student;
   }
}
```

Figure 5.7: Service logic of student Web Service

Using the approach, the java source codes have been implemented before writing the Web Services description. The example of the student Web Service is used to retrieve student information by passing an argument of a student ID. Data Access Object (DAO) is adopted to map a database object to a persistence object. One of the advantages of

DAO is that it can provide data operations without exposing the details of the database, for instance, the database structure, primary key or foreign key defined in the tables, the one-to-many or many-to-one relationships between tables. The Eclipse IDE platform can generate a Web Service and a client without writing any source code. A screenshot of generating the student Web Service and the client using Eclipse is shown in Figure 5.8 and Figure 5.9. All the source codes of the student Web Service are presented in Appendix A.

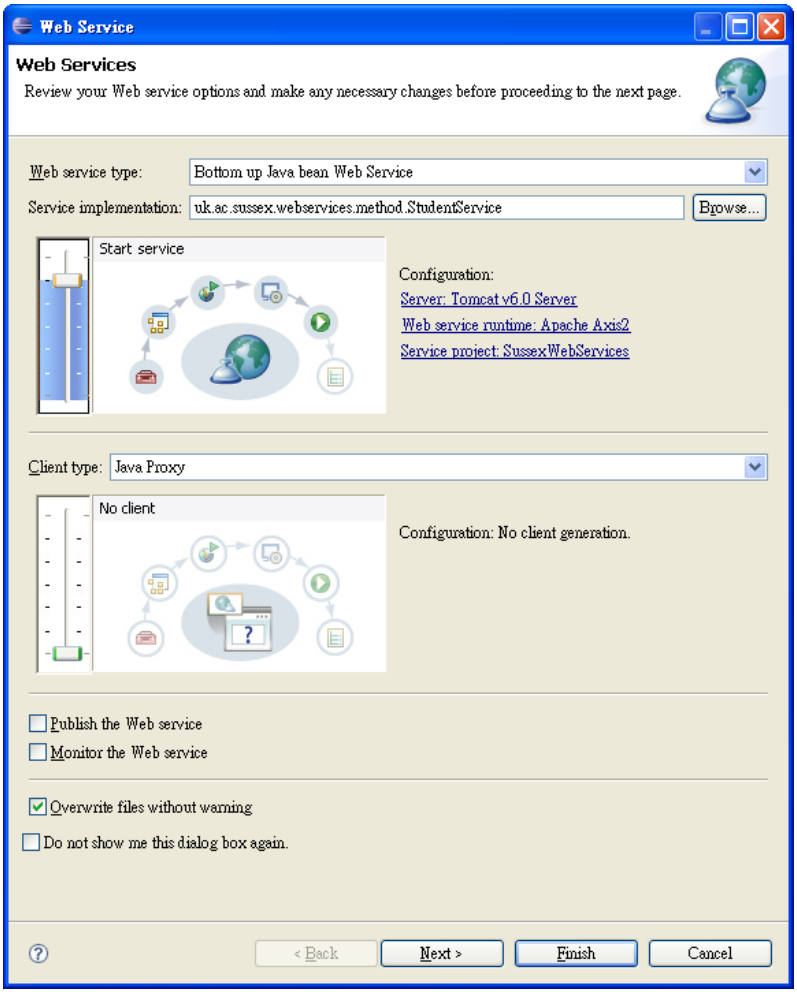

Figure 5.8: Generate a student Web Service using Eclipse

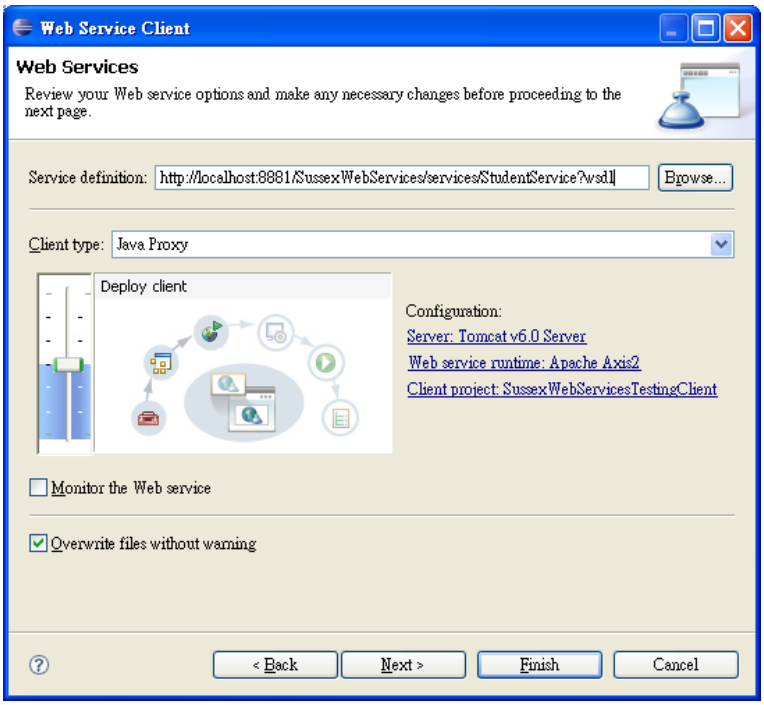

Figure 5.9: Generate a student Web Service client using Eclipse

Using the Eclipse Web Services feature, all the required XML files and Web Service Description Language (WSDL) have been generated, such as StudentService.wsdl. Moreover, the student Web Service can be tested by a testing client that is also generated by Eclipse as shown in Figure 5.9. A testing client can be a Web application or a standalone application. In order to show the SOAP messages between sender and receiver when invoking the student Web Service, the Eclipse Web Services Explore is used as shown in Figure 5.10. It can monitor both directions of SOAP messages including a Web Service request and response messages. In Figure 5.11, there is a SOAP envelope with a simple body element to invoke the student Web Service by calling the method name “getStudent” with an argument “studentId” and the value is set to 1. The Tomcat Web container uses the Axis2 plug-ins to parse, process and construct a SOAP response message as shown in Figure 5.12. This response message is simple and uses the request tag “getStudent” plus “Return” label to return the result.

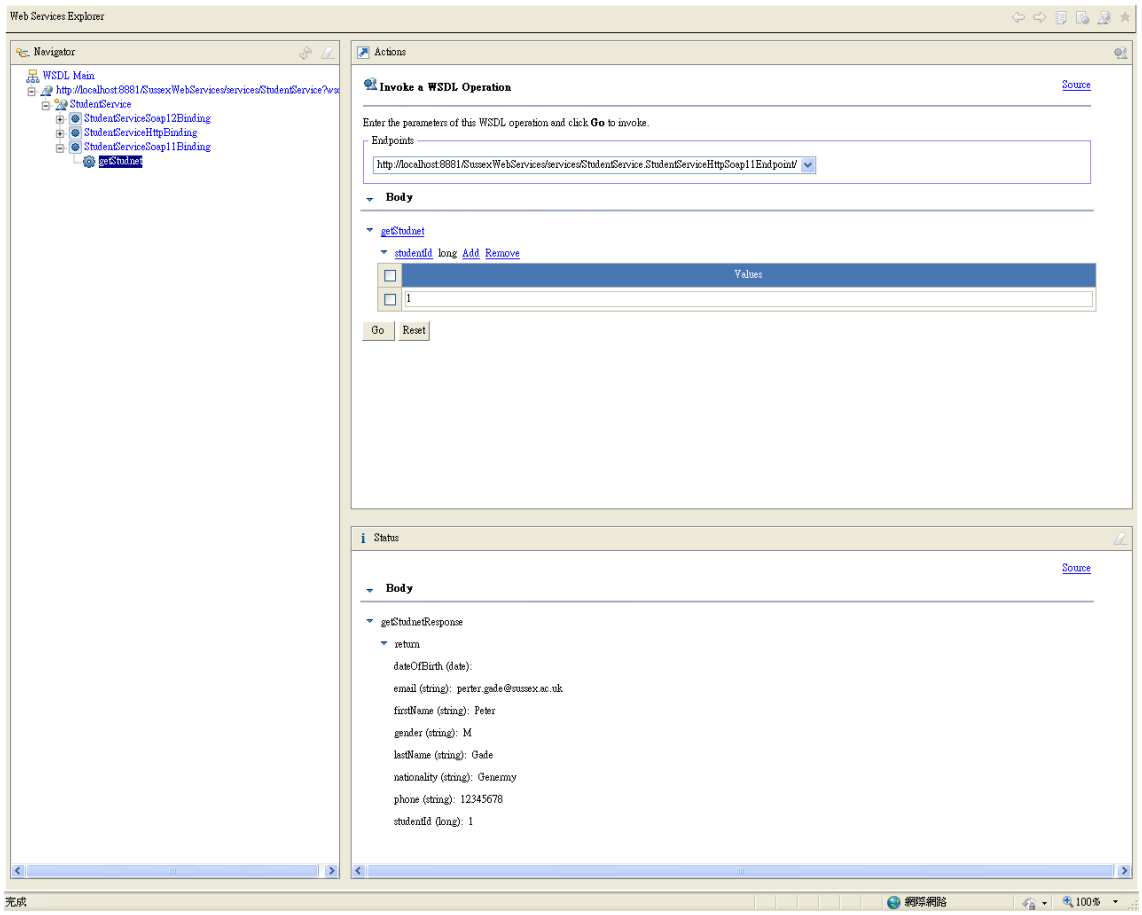

Figure 5.10: Using Eclipse Web Services Explore to test the student Web Service

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:q0="http://method.webservices.sussex.ac.uk"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <soapenv:Body>

    <q0:getStudnet>

      <q0:studentId>1</q0:studentId>

    </q0:getStudnet>

  </soapenv:Body>

</soapenv:Envelope>
```

Figure 5.11: A SOAP request message for invoking the student Web Service

```
<?xml version="1.0" encoding="http://schemas.xmlsoap.org/soap/envelope/"
standalone="no"?>

<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">

<soapenv:Body>

<ns:getStudnetResponse xmlns:ns="http://method.webservices.sussex.ac.uk">

<ns:return xmlns:ax21="http://entity.sussex.ac.uk/xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="ax21:Student">

<ax21:dateOfBirth xsi:nil="true"/>

<ax21:email>perter.gade@sussex.ac.uk</ax21:email>

<ax21:firstName>Peter</ax21:firstName>

<ax21:gender>M</ax21:gender>

<ax21:lastName>Gade</ax21:lastName>

<ax21:nationality>Genermy</ax21:nationality>

<ax21:phone>12345678</ax21:phone>

<ax21:studentId>1</ax21:studentId>

</ns:return>

</ns:getStudnetResponse>

</soapenv:Body>

</soapenv:Envelope>
```

Figure 5.12: A SOAP response message after invoking the student Web Service

In this section, the Eclipse Integrated Development Environment (IDE) and Java Platform are adopted to demonstrate how to create a student Web Service and a stand-alone Web Service application. The communication traffic or the SOAP request and response messages can also be monitored by using the Eclipse Web Serves Explorer. However, these are not the only tool to implement Web Services. Microsoft .Net, IBM developerWorks, WebLogic, etc. can also be used to develop and deploy the Web Services.

5.5 Web Services Security Libraries

Security is one of the major requirements for many types of computer systems, especially for exchange of data over the Internet. Traditional information systems have moved from the client-server model to Web Services. The implementation of Web Services has been designed and developed, for instance, Apache Axis, Apache Axis2, JAX-WS (Java API for XML Web Services), Microsoft .NET Framework, etc. Apache Axis2 is a Web Services / SOAP / WSDL engine. It can send, receive and parse a SOAP message. There are two layers of Web Services security. The first one is a transitional Web security mechanism, which is done in the transport layer using SSL or TSL to protect the communication path between sender and receiver. It is the point-to-point protection. The second one is implemented in the message layer which applies to XML documents that are sent as SOAP messages, it provides message confidentiality, prevention of message alteration, etc. It is the end-to-end protection. To support secure Web Services applications at the message level, the WS-Security standards or specification are adopted. The WS-Security API for implementing the WS-Security core specification and secure token profiles have been developed. There is a famous toolkit which is used to implement the WS-Security specification, which is Apache WSS4J. The Apache WSS4J project is one of the Java implementations of the primary security standards for Web Services and it supports the following WS-Security standards:

- SOAP Message Security 1.1
- Username Token Profile 1.1
- X.509 Certificate Token Profile 1.1
- SAML Token Profile 1.1
- Kerberos Token Profile 1.1
- Basic Security Profile 1.1

Apache WSS4J is part of the Apache Web Services project and it provides a set of APIs to implement WS-Security functionality of a SOAP message. The WSS4J is an implementation of the OASIS Web Services Security specifications, which is a Java library that can be used to sign, verify, encrypt and decrypt a SOAP message. This library is independent and can use the WSS4J's APIs directly in a standalone manner.

All the related classes and interfaces can be found in the org.apache.ws.security package. It can be used as a library to sign and verify parts of, or the entire SOAP message and it also interoperable with Java API for XML-based RPC (JAX-RPC) and Microsoft .NET-based servers and clients. Using the WSS4J APIs, a Web Service can add WS-Security functionality to enable the authenticity, integrity and non-repudiation.

However, the Apache WSS4J package is not adopted in this research because the performance evaluation among the secure token profiles and proposed token needs to be compared in a fair situation. The processing time or resources for parsing and processing the secure token profiles will be evaluated in the pure version. Therefore, the implementation of the secure token profiles is rewritten and the Participant Domain Name Token is designed and programmed.

5.6 Implementation of Participant Domain Name Token

Web Services uses SOAP messages to represent remote procedure calls between client and server. Therefore, a SOAP message parser is needed. The SOAP message parser is software that reads a SOAP message and obtains the information from the message, which will be manipulated in a program, for instance, an XML parser converts an XML document into an XML DOM object, which can be manipulated in a Java program. SOAP with Attachments API for Java (SAAJ) is used for mainly for the SOAP messaging that goes on behind the scenes in JAX-RPC and JAXR implementations. The SAAJ is adopted in the performance evaluation because it can read and write SOAP messaging directly rather than use JAX-RPC. The SAAJ API conforms to the Simple Object Access Protocol (SOAP) 1.1 specification and is implemented in the java.xml.soap package. The SAAJ-related classes are located in the java.xml.soap package and this package has all the APIs necessary for sending requests and responding to a SOAP message. The Participant Domain Name Token profile is implemented in Java language and uses “java.xml.soap.*” Java Achieve (JAR) to parse a SOAP message. Other libraries such as “org.xbill.DNS.*” are used for SRV and address resource record resolution. All the source codes of the PDNT implementation are presented in Appendix B.

5.6.1 The java.xml.soap Package

A SOAP message can be generated and sent manually but the SOAP with the attachments API for Java (SAAJ) automates many of the required steps. The package javax.xml.soap is a Java API for parsing, creating and building a SOAP message and it extends their counterparts in the org.w3c.dom package. This package is defined in the SOAP with attachments API for the Java (SAAJ) 1.3 specification. According to the Java™ Platform, Standard Edition 6 API specification, these package facilities the following:

- create a point-to-point connection to a specified endpoint
- create a SOAP message
- create an XML fragment
- add content to the header of a SOAP message
- add content to the body of a SOAP message
- create attachment parts and add content to them
- access/add/modify parts of a SOAP message
- create/add/modify SOAP fault information
- extract content from a SOAP message
- send a SOAP request-response message

The java.xml.soap package extends DOM API to manipulate a SOAP message and builds up a SAAJ tree. It is possible to use DOM APIs to add ordinary DOM nodes to a SAAJ tree. However, the SAAJ APIs are still required to return SAAJ types when examining and manipulating the tree. The interface of SAAJ is used for SOAP messaging that provides a way to send XML documents in SOAP format over Internet from a Java programming model. The class and interface hierarchy of package java.xml.soap are shown in Figure 5.13 and 5.14. A Web Service application can send a SOAP message directly using an object of SOAPConnection. A SOAP message object can be created by using the MessageFactory object. The message object which is created by MessageFactory contains the basic parts of a SOAP message including SOAPEnvelop, SOAPHeader, SOAPBody, etc. Therefore, a Web Services application can manipulate, send the SOAP message by use of these objects.

- java.lang.Object
  - javax.xml.soap.AttachmentPart
    - javax.xml.transform.dom.DOMResult (implements javax.xml.transform.Result)
      - javax.xml.soap.SAAJResult
  - javax.xml.soap.MessageFactory
  - javax.xml.soap.MimeHeader
  - javax.xml.soap.MimeHeaders
  - javax.xml.soap.SAAJMetaFactory
  - javax.xml.soap.SOAPConnection
  - javax.xml.soap.SOAPConnectionFactory
  - javax.xml.soap.SOAPElementFactory
  - javax.xml.soap.SOAPFactory
  - javax.xml.soap.SOAPMessage
  - javax.xml.soap.SOAPPart (implements org.w3c.dom.Document, javax.xml.soap.Node)
  - java.lang.Throwable (implements java.io.Serializable)
    - java.lang.Exception
      - javax.xml.soap.SOAPException

Figure 5.13: Class hierarchy of package java.xml.soap

- javax.xml.soap.Name
- org.w3c.dom.Node
  - org.w3c.dom.CharacterData
    - org.w3c.dom.Text
      - javax.xml.soap.Text (also extends javax.xml.soap.Node)
  - org.w3c.dom.Element
    - javax.xml.soap.SOAPElement (also extends javax.xml.soap.Node)
      - javax.xml.soap.DetailEntry
      - javax.xml.soap.SOAPBody
      - javax.xml.soap.SOAPBodyElement
        - javax.xml.soap.SOAPFault
      - javax.xml.soap.SOAPEnvelope
      - javax.xml.soap.SOAPFaultElement
        - javax.xml.soap.Detail
      - javax.xml.soap.SOAPHeader
      - javax.xml.soap.SOAPHeaderElement
  - javax.xml.soap.Node
    - javax.xml.soap.SOAPElement (also extends org.w3c.dom.Element)
      - javax.xml.soap.DetailEntry
      - javax.xml.soap.SOAPBody
      - javax.xml.soap.SOAPBodyElement
        - javax.xml.soap.SOAPFault
      - javax.xml.soap.SOAPEnvelope
      - javax.xml.soap.SOAPFaultElement
        - javax.xml.soap.Detail
      - javax.xml.soap.SOAPHeader
      - javax.xml.soap.SOAPHeaderElement
    - javax.xml.soap.Text (also extends org.w3c.dom.Text)
- javax.xml.soap.SOAPConstants

Figure 5.14: Interface hierarchy of package java.xml.soap

To create and send a SOAP message using SAAJ, five steps are required including:

- Creating a SOAP connection
- Creating a SOAP message

- Populating the message
- Sending the message
- Retrieving the reply

An example of using SAAJ is shown in Figure 5.15. The object of SOAPConnection is used to make a one-way trip from one endpoint to another endpoint and uses HTTP Post as a transport mechanism. After creating a SOAPConnection object, a SOAP message object can be created by invoking createMessage() method, which is defined in a MessageFactory class. The SOAP message object has four elements, which are SOAPPart object, SOAPEnvelope object, SOAPBody object and SOAPHeader object. The four objects can be retrieved from a SOAPMessage object by use of the corresponding getter, such as using getBody() method to obtain a SOAPBody object. An empty SOAPBody object will be generated when a new SOAP message object is created. To manipulate or hold the content of a SOAPbody object, a SOAPBodyElement object and Name object are used. The method createName() and addBodyElement() of Name object and SOAPBodyElement objects are used to populate a SOAP message. After creating and populating a SOAP message, the message can be sent using a SOAPConnection object by invoking the call() method with an argument of a URL object. The call blocks until it receives a response message. A response message will be sent back from another endpoint, the response must be a SOAPMessage object. Therefore, the response message content can be accessed using the same methods as those for giving content to a message. The SOAPObody object can be acquired through the SOAPMessage, SOAPPart or SOAPEnvelope Objects.

```
public static void main(String args[]) {

   try {

      //Create a SOAPConnection

      SOAPConnectionFactory factory = SOAPConnectionFactory.newInstance();

      SOAPConnection connection = factory.createConnection();

      //Create a SOAPMessage

      SOAPMessageFactory messageFactory = MessageFactory.newInstance();

      SOAPMessage message = messageFactory.createMessage();

      SOAPPart  soapPart = messge.getSOAPPart();

      SOAPEnvelope envelope = soapPart.getEnvelope();

      SOAPHeader header = envelope.getHeader();

      SOAPBody body = envelope.getBody();

      header.detachNode();

      //Populate a SOAPMessage

      Name bodyName = envelope.createName("GetStudentInfo","m",
      "http://webservice.sussex.ac.uk");

      SOAPBodyElement bodyElement = body.addBodyElement(bodyName);

      Name name = envelope.createName("symbol");

      SOAPElement symbol = bodyElement.addChildElement(name);

      symbol.addTextNode("studentID");

      //Sending a SOAPMessage

      URL endpoint = new URL("http://webservice.sussex.ac.uk");

      SOAMessage response= connection.call(message, endpoint);

      // Close the SOAPConnection

      connection.close();

      } catch (Exception e) {

         System.out.println(e.getMessage());

      }

  }

}
```

Figure 5.15: An Example of creating and sending a SOAP message using SAAJ

5.6.2 The org.xbill.DNS Package

The proposed token profile uses a service record to authenticate a message sender. The service record is one of the DNS records and can be retrieved by a DNS query. The org.xbill.DNS or “dnsjava” package supports all record types defined in RFC 1035,

RFC 3596, RFC 2782, etc and is implemented in the Java programming language. There are many applications using this package. Examples are shown in the following:

- CustomDNS – a customize DNS server written in Java
- CRSMail – a Java based E-Mail server
- Rabbit4 – a Web proxy server
- Eagle DNS – an authoritative DNS server

To implement the Participant Domain Name Token Profile, the org.xbill.DNS APIs are used to retrieve the Service Record (SRV), which is stored in a DNS server. For instance, a Web Service handler parses a SOAP message to construct a DOM tree and obtains a participant domain name element, < pdn:ParticipantDominName > from the tree. Then, the org.bill.DNS library will be invoked to acquire the relative Service Record and Address Record. Figure 5.16 shows an example of a Java method using the dnsjava package.

```
public List<String> getSrvRecord(String lookupService){
      List<String> hostList=new ArrayList<String>();
      Record [] records = new Lookup(lookupService, Type.SRV).run();
      if (records!=null){
         for (int i = 0; i < records.length; i++) {
            SRVRecord srvRecord=(SRVRecord) records[i];
            hostList.add(srvRecord.getTarget().toString());
         }
      }
      return hostList;
}
```

Figure 5.16: A Java method uses org.bill.DNS to retrieve a list of Service Records

5.6.3 The Participant Domain Name Package

A new java class to handle the proposed token named "ParticipantDomainNameTokenHandler" has been designed and developed. A Java class constructor and a validation method of the proposed token are shown in Figure 5.17 and an example of usage is shown in Figure 5.18. An instance, which is created from the class of the ParticipantDomainNameToken (PDNT) has been tested on different sizes of SOAP messages and it can work with other secure token profiles. All the related classes and libraries are packed into a JAR file, which can be used and plugged into different Web Services containers.

To instance the PDNT class, a SOAPMessage object is an argument or parameter for creating the PDNT object. The PDNT will access the content of a SOAP message and validate the domain name by calling the validate() method. The PDNT object accesses the content of a SOAPMessage by using the SAAJ package. The TextContent of the "pdn:ParticipantDomainName" and "pdn:DomainName" elements are obtained by calling getElementsbyTagName method. The service record can be found within the pdn:DomainName element. It is used to find out the corresponding address record and compares it the IP address that the Web container shows.

The ParticipantDoaminNameToken (PDNT) is designed and developed in a standalone Java package. It means that it can be plugged into different J2EE Web containers or other Java libraries to implement or adopt the proposed token profile. All the related java class files, associated metadata and resources are aggregated into one Java Archive (JAR) file. The JAR file allows Java runtime to efficiently deploy a set of classes and their associated resources.

```
public ParticipantDomainNameTokenHandler(SOAPMessage message){
   try{
      soapMessage=message;
      this.soapPart = soapMessage.getSOAPPart();
      this.soapEnvelope = soapPart.getEnvelope();
      this.soapHeader = soapEnvelope.getHeader();
      this.soapBody=soapEnvelope.getBody();
      if (soapHeader==null){
         soapHeader=
         MessageUtil.createSecureHeader(soapEnvelope);         }
   }catch(Exception e){
      e.printStackTrace();
   }
}

public boolean validate(String senderIP){
   String lookupService=getLookupService();
   List <String> ipList=DNSClient.getAddressRecord(lookupService);
   for (int i=0;i<ipList.size();i++){
      if (ipList.get(i).equals(senderIP))
         return true;
   }
   return false;
}
```

Figure 5.17: The class constructor and a validation method of Participant Domain Name Token

```
public static void main(String[] args) {
   MessageFactory factory = MessageFactory.newInstance();
   MimeHeaders mimeHeaders = new MimeHeaders();
   mimeHeaders.addHeader("Content-Type","text/xml;charset=UTF-8");
   SOAPMessagesoapMessage=factory.createMessage(mimeHeaders,
   MessageUtil.readFileToInputStream("soapMessage.xml"));
   ParticipantDomainNameTokenHandler tokenHandler=new
   ParticipantDomainNameTokenHandler(soapMessage);
   tokenHandler.validate(senderIP);
```

Figure 5.18: A sample program using Participant Domain Name Token

5.7 Conclusion

This chapter presented the detailed implemention methodologies to develop the PDNT. Because of the security issues with existing Web Service security standards, PDNT is proposed and it has performance advantages that have been demonstrated by the performance evaluation. The research uses Java as the major programming language to implement the PDNT. Moreover, the Attachments API for Java (SAAJ) package or toolkit is used for parsing a SOAP message. In order to work with DNS resource records, the “dnsjava” package is also adopted. This package supports all record types of the DNS resource record and it is implemented in the Java programming language. All the classes to handle and process the PDNT were designed and developed. Moreover, they were collected as a standalone Java package and aggregated into one Java Archive (JAR) file. Therefore, it can be plugged into J2EE Web containers or become a Java library for other systems use.

CHAPTER 6

– GENERAL DISCUSSION, CONCLUSION AND FUTURE DIRECTIONS

6.1 General Discussion

An Organization must adapt to the changes in both external and internal environments both to survive and gain the best return on investment. These changes will affect the direction of business strategy, partnerships, mergers and acquisitions. The IT architecture and systems must also evolve to fulfill the new business environment and requirements, for instance, integration across enterprise boundaries, system collaboration between partners, customers and suppliers. Therefore, the IT architectures and systems must be designed to be flexible to reflect these changes. Service-Oriented Architecture (SOA) is an IT architectural style and composed of loosely coupled invokable software modules or services. Service-Oriented Architecture (SOA) has gained widespread acceptance in different sectors. It allows reuse and integration of existing IT assets, for instance: business logic reused and heterogeneous systems integration. The concept of the SOA can be used in other frameworks, for instance, Enterprise Service Bus (ESB) and Enterprise Application Integration (EAI). The ESB is a messaging backbone to do the data conversion, format transformation, routing, accept and deliver messages. The EAI is a framework to integrate a set of applications within or across the enterprises. These frameworks can be developed or implemented by a set of services and then they also become the Service-Oriented Architecture.

The SOA is a preferred design and architecture style for integration of ubiquitous computing resources. Traditionally designed ICT systems use an application style, which is designed as a tightly coupled system; this is less effective and suffers from the dependencies of each component. The invention of Service-Oriented Architecture (SOA) makes each component system independent and permits a loosely coupled system. The SOA can be adopted in a large complex system which includes many independent components. For instance, online shopping applications are composed of different functionalities like credit card authorization, currency conversion, best price searching, etc. These components can be designed and implemented into several independent services. The services can then be used in a single application or other applications.

Services reuse is one of the advantages of adopting SOA. Moreover, adopting the SOA can gain benefits in enterprise application integration, service reuse, leveraging the legacy investment and best of breed integration. Therefore, SOA is suitable to design a distributed, Internet-based, dynamic change, autonomous and non-point to point system.

The SOA can be implemented by a set of independent software units or services, with interfaces that can be invoked to perform required tasks. It can be implemented by SOAP-based Web Services, RESTful Web Services, Remote Procedure Call (RPC), Distributed Component Object Model (DCOM), Common Object Request Broker Architecture (CORBA) etc. However, Web Services are the most popular technology to implement the SOA. A Web service is a kind of distributed system that provides APIs, which can be used by all systems on the Internet. Unlike a traditional Application Programming Interface (API), Web Services API is language-independent and can be invoked by different languages on different platforms. There are two major classes of Web Services: REST-compliant Web Services and SOAP-based Web Services. Both have their advantages and disadvantages. The RESTful is simple and flexible but the SOAP is a standard protocol, which includes more extensions and security mechanisms.

REST-compliant Web Service is also called RESTful Web Services. It is not a standard and the message consists of XML and HTML. RESTful Web Services are a resource-oriented architecture and uses Unique Resource Identifier (URI) to represent each object in the system. RESTful focuses on interfacing with resources and changing their state. It uses a directory structure-like URIs to show where a service consumer can acquire the resources. Unlike other Web Services architecture, it works on resources directly rather than invokes a function to work on resources. RESTful Web Services are stateless; the state is maintained by being transferred from a client to a server and back to the client. It uses a uniform set of stateless operations to parse and process a Web Service request. A RESTful Web API is a Web API that utilizes HTTP and REST principles. All interfaces are limited to the four HTTP commands. It makes explicit use of the HTTP methods, which include GET, POST, PUT and DELETE to implement read, create, update and delete functions (CRUD). RESTful Web Services do not use RPC to send a message

Unlike RESTful, the SOAP-based Web Service not only supports HTTP protocol but also SMTP, FTP and other communication protocols. Although RESTful Web Services are simple and flexible compared with SOAP-based Web Services or stateful Web Services, it does not provide any security mechanism to protect a message. In order to protect the message content, HTTPS is used to secure the transmission of the message over the network if the RESTful style is adopted. However, it does not provide end-to-end protection. SOAP-based Web Services is highly extensible and it can cooperate with many existing standards and protocols. WS-Security is a set of security standards for SOAP-based Web Services which offers message confidentiality and message integrity. It also provides user authentication, authorization and other security mechanisms. Moreover, the state of a SOAP-based Web Service is maintained on the server side. It means that state information is not kept on the client side even when the session is still alive. Therefore, Simple Object Access Protocol is the most common way to implement Web Services if security is a major concern. The SOAP is a specification for exchanging structured messages between two end points via common communication protocols such as Hypertext Transfer Protocol (HTTP). The major security challenges between two end points are in the transport layer and message layer. The transport level security uses layer 3 and layer 4 protocols to protect data, using IPSec, SSL, etc. The message layer security is a proper method to protect the message format from end-to-end and it operates on layer 7. The SOAP-based Web Services has many in-built security mechanisms and extensions running on layer 7. It is a suitable choice if the security is a major concern for the SOA-based system.

An example of a SOA-based system, which requires higher security is presented in this research, named “An SOA-based Disease Notification System”. Disease notification is one of the critical components for prevention and controlling the spread of infectious disease within society and around the world. The proposed system uses medical standards, for instance, International Classification Code (ICD) to exchange a notifiable disease through the Internet. Three layers are used in the proposed system including the service layer, business process layer and connectivity layer. In the service layer, a service is a fundamental component of the Web Services. The business logics are

decomposed into several generic services. All the services can interact with each other at the message level in the business process layer and the execution order can be specified and the business logic of inter-services programmed using the Business Process Execution Language (BPEL). A precise and reliable laboratory test result can indicate whether it is a notifiable disease. In a heterogeneous laboratory environment, a laboratory test result is generated by an analyzer or a medical instrument. The results that are printed out from the Laboratory Information Management System (LIMS) or input to a health information system (HIS) manually for further investigation. However, there are many different types of medical instruments in a hospital environment and each of them produces medical results in different formats. In order expedite speed of response issues; the Enterprise Service Bus (ESB) is used in the connectivity layer to integrate everything together. The ESB does not only interface to all clinical instruments, but also interfaces to LIMS and HIS. Therefore, the precise and reliable laboratory test results can be delivered to the HIS on time, which is important during the outbreak of disease.

A service provider deploys a Web Service for a service consumer to submit notifiable diseases, for instance, a notifiable disease will be sent to a local health authority directly when a hospital invokes a Web Service of local health authority or Centers for Disease Control (CDC). The hospital can also act as a service provider to provide a Web Service for a clinic to submit a notifiable disease to them. All the notifiable diseases can also be sent to the World Health Organization (WHO) from different health authorities via a Web Service, which is provided by the WHO. Existing WS-Security standards can control who can submit a notifiable disease. However, it cannot handle where can submit a notifiable disease. Location authentication is important for disease notification, for instance, it does not allow submission of a SARS case outside of a hospital. Additional authentication mechanisms can ensure case authenticity.

Security is important for the use of Web Services. However, performance is another critical factor for evaluating a new security standard. Performance measurements have been completed in this research to compare the existing secure token profiles, the WS-

Security with the proposed token profile. Response time or latency time is adopted as a performance metric to evaluate the round-trip time of a message between sender and receiver. The latency time is affected by two factors, which are the total time spent in the network layer and application layer. However, the total time spent in the network layer is not considered because it is difficult to evaluate over the Internet. The total time spent in the application layer is also affected by many factors including message size, number of XML elements, parsing time, token handling time, total time spent in business logic and database manipulation, etc. In this performance evaluation, all the measurements are made with identical equipment and environment. The proposed token requires a DNS lookup to acquire an SRV record for the domain validation. However it is not a time consuming process because a local DNS cache is used after first time querying. Therefore, the network issue and DNS lookup time can be ignored and only latency time spent in the application layer is considered. A millisecond timescale is used to compute latency for each round-trip time between message sender and receiver. Both the Web Service provider and consumer are written in the Java programming language. A pure HTTP server and HTTP client with necessary java library are used for the performance evaluation.

Different sizes of SOAP messages were used for each test case and average latency time is adopted for comparison between the proposed token and WS-Security tokens. The size of a SOAP message depends on which secure tokens are used. According to the message size evaluation of each secure token profile in this research, the message size of the proposed token profile is the minimum and the encryption token profile is the maximum, which is increased by 37%. Four test cases were selected and are tested with different message sizes, which include the proposed token vs. (1) Username token; (2) XML encryption token; (3) XML signature token; (4) XML encryption with signature token. In each test case, the proposed token profile was used by a message receiver to authenticate the location of a sender. WS-Security token profiles were used to validate a SOAP message by different mechanisms. Each test case has been divided into three sub-test cases, which was processed a hundred times to obtain the average latency for each sub-test case of each record size.

In the test case 1, the latency of processing Participant Domain Name Token (PDNT) is 12.74 % faster than username token profile when a message contains 100 employee records. It is also 6.67%, 2.27%, 1.36% and 0.07% faster if adopted for 200, 300, 400 and 500 employee records respectively. The encryption token profile uses the most processing time when compared with the other WS-Security secure token profiles. The results are shown in test case 2. It is 238.13 % ~ 843.5% faster than the encryption token profile when the proposed token is adopted. In test case 3, the latency of processing PDNT is 30.3% ~ 50.39% faster than signature token profile. Finally, the encryption token with signature token are selected to compare with the proposed token for the performance evaluation and the results are shown in test case 4. It shows that the PDNT is 257.12% ~ 884.62% faster than these two token profiles. Therefore, all test case results show that the processing time for adopting the proposed token profile is faster than the other secure token profiles. The four test cases also evaluate the overhead to process the proposed token with each WS-Security token profile. In test case 1, if both the proposed token and username token are used and processed, the processing time is increased by 0.37%. It means that by adopting an additional token, the proposed token there is a very small increase and it provides additional security. In test case 1, 2 and 3, it shows that the processing time is increased by less than 1% to process the proposed token with encryption token, proposed token with signature token, proposed token with encryption and signature token. Therefore, the overhead is not significant and the system gains an additional security feature if using PDNT.

The security of an IT system has become one of the most important components of the system, especially for Internet-based systems. In an organization, the primary security device is a firewall and a SOAP message is designed to move across firewalls by using HTTP protocol. However, most firewalls are a layer 3 security device and cannot provide end-to-end message authentication, authorization, confidentiality and integrity. In order to protect the content or data in a message, more than one security feature is used in a single SOAP message. However, the more security mechanisms that are adopted the greater will be the total processing time. In order to gain a performance advantage the proposed token eliminates the overhead for processing an unknown or a fake request, the PDNT is processed before other WS-Security secure tokens, such as

username token, encryption token, signature token, and etc. Therefore, it can refuse a fake request as fast as possible and does not need to process other secure tokens if it is an illegal request, which will saves the processing resources to handle more requests.

In this research, message layer security has been taken into account in order to fulfill three major aspects of security, integrity, confidentiality and availability. The SOAP relies on XML for its message format and a SOAP message can be protected in the message layer. We found existing message layer security mechanisms, which are defined and collected in the OASIS standard 1.1. It does not only provide message integrity and message confidentiality but also other secure token profiles including:

- Username Token Profile
- X.509 Token Profile
- SAML Token Profile
- Kerberos Token Profile
- Rights Expression Language Token Profile

Each token profile has defined a standard set of Web Services Security extensions to profile particular security features. We found that the five secure token profiles focus on two areas. Who can use the Web Services and what are the permissions. However, we found that the location of a Web Service invoker is not handled and not verified in existing secure token profiles. Other researchers also focus on the dynamic policy assignment based on other techniques, for instance, data mining and ontology. The location of service invoker is also ignored. Therefore, a location based verification is needed to control “where” can use the Web Services.

6.2 Conclusions

Existing standards on WS-Security and performance evaluation of the proposed token have been presented in this thesis. The features, purposes and limitations of the OASIS standard 1.1 have also been discussed. The research have found and achieved the following:

1. Services-Oriented Architecture (SOA) is one of the preferred approaches for system design and system integration, since integrated computing has become ubiquitous.
2. Some SOA-based systems require higher security, for instance, the disease notification system.
3. RESTful Web Services and SOAP-based Web Services are the widely used technologies to implement Web Services.
4. Web Services is one of the technologies to implement SOA and Simple Object Access Protocol (SOAP) is widely used to implement Web Services if security is a major concern.
5. SOAP-based Web Services is an XML-based platform-independent protocol. It is used for data exchange, method discovery and invocation between Web Services applications.
6. WS-Security 1.1 OASIS is a security standard and specification to provide message confidentiality, message integrity, user authentication, authorization and other security mechanisms for SOAP-based Web Services.
7. The SOAP-based message can be protected in both the transport and message layer protocol. Transport layer protection can be achieved by standard network security protocol, for instance SSL, IPSec. The message layer is a proper method to protect the message from one end to the other end. WS-Security 1.1 OASIS is an existing standard to protect a SOAP-based message, which provides “who can use the services” and “what are the permissions” mechanisms.
8. There are no location-based mechanisms for the location authentication using existing standards and research.

A location-based authentication mechanism is proposed in this research, named Participant Domain Name Token (PDNT). “Where can use the services” is handled by the new proposed token and it is implemented in the Java language. Location authentication is important for a SOA-based system which is required to monitor and control “where” can invoke the Web Services. The Disease Notification System is an example of a system which requires control of “where” can submit a notifiable disease to ensure case authenticity. To adopt PDNT, the following libraries are required:

1. Java.xml.soap.* are used to parse a SOAP message.
2. Org.xbill.DNS.* are used for SRV and address resource record resolution.

All the relative classes are packed into a Java Achieve (JAR) file. A web container or a Web Services system can plug this jar file into existing system architecture with minor modification. Performance is another advantage to be gained by adopting the PDNT. Based on the experimental results, which are presented in this thesis, the PDNT gains significant performance advantage over other secure token profiles and the overhead for adopting PDNT is minor. Therefore, all the participants can reap the benefits of increased security and performance.

The proposed token provides a location-based authentication mechanism. It is not only used to control an area but also control a country and region to invoke a service. The PDNT can disallow a region that cannot invoke the Web Services, even though a user has permission. Therefore, this new security token could have a significant impact in combating industrial espionage by commercial organizations and National Security Agencies.

6.3 Future Directions

OASIS Web Services Security standards do not only enhance a SOAP message to provide message integrity and confidentiality but also define how to use the secure token to enable the implementation of a wide range of protocols, such as Kerberos,

X.509. The proposed token enhances the WS-Security by adding a new secure token to provide location-based authentication mechanisms. It also respects the WS-Security specification, which fulfills the requirements to use the SOAP header for carrying security information. The PDNT can be used for the authentication of a message sender location. The newly designed token can be used to reject invalid requests or responses, which come from unknown or fake domains before parsing and processing the other secure token profiles.

The proposed secure token profile incorporates one of the most widely used Internet service information identities, which is the Domain Name Service (DNS). It can verify, control and monitor the location of the service consumer or message sender. An existing DNS record is used to describe or show participant domain information, which is Service Record (SRV), a DNS resource record for specifying the location of services. However, the target field defined in the SRV record has been a little changed since the original meaning to the self-defined. Therefore, it will confuse a user who follows the SRV specification, the RFC2782.

In the future, a new type of DNS resource record should be considered to support the proposed token. This record can be used to properly support the PDNT without changing the usage of the SRV record. The Network Address Translation (NAT) is also considered if a participant uses a firewall between internal network and Internet. One-to-many NAT is usually adopted for multiple private hosts mapped to a public IP address. If there are many departments within an organization invoking Web Services, which are outside the organization, for instance, on the Internet, only a single public IP can be used to validate all departments by the PDNT handler because of the NAT issue. In the future, we will provide mechanisms to tackle this issue. Moreover, the PDNT should be implemented in a real service consumer and service provider and become a de facto Web Services secure token profile.

7

- LIST OF JOURNAL AND CONFERENCE PAPERS PUBLISHED

2012

Chi Po Cheong, Chris Chatwin, Rupert Young, "Performance Enhancement of WS-Security Using Participant Domain Name (PDNT)", 2012 9th International Joint Conference on Computer Science and Software Engineering (JCSSE2012), 30 May – 1 June 2012, Bangkok, Thailand, pp. 214-219, ISBN 978-1-4673-1920-1, Won Best Paper Award-Double Blind reviewed

Chi Po Cheong, Simon Fong, Pouwan Lei, Chris Chatwin, Rupert Young, Designing an Efficient and Secure Credit Card-based Payment System based on ANSI X9.59-2006 with Web Services, Journal of Information Processing System, Korea, Information Processing Society, Volume 8, Issue 3, 2012, pp. 495-520, ISSN: 1976-913X (Print), ISSN: 2092-805X (Online)

2011

Chi Po Cheong, Chris Chatwin, Rupert Young, "A New Secure Token For Enhancing Web Service Security", 2011 IEEE International Conference on Computer Science and Automation Engineering (CASE 2011), 10-12 June 2012, Shanghai, China, Volume 1, pp. 45-48, ISBN 978-1-4244-8727-1

2010

Chi Po Cheong, Chris Chatwin, Rupert Young, "A Framework for Consolidating Laboratory Data Using Enterprise Service Bus", 2010 3rd IEEE International Conference on Computer Science and Information Technology, 9-11 July 2010, Chengdu, China, pp. 557-560, ISBN 978-1-4244-5537-9

2009

Chi Po Cheong, Chris Chatwin, Rupert Young, "An SOA-based Disease Notification System", 7th International Conference on Information, Communications and Signal Processing (ICICS 2009), 7-10 December 2009, Macau, China, pp. 1-4, ISBN 978-1-4244-4656-8

Chi Po Cheong, Chris Chatwin, Rupert Young, "A RDF-based Semantic Schema Mapping Transformation System for Localized Data Integration", 3rd International Conference on Anti-counterfeiting, Security, and Identification (ASID 2009), 20-22 August 2009, Hong Kong, China, pp. 144-147, ISBN 978-1-4244-3883-9

8
– BIBLIOGRAPHY

Addie R. G., Moffatt S., Dekeyser S., "Five examples of web-services for illustrating requirements for security architecture", 2011 International Conference on Data and Knowledge Engineering, 6 September 2011, Milan, Italy, 2011, pp. 47-54. ISBN 978-1-4577-0865-7

Albreshne A., Fuhrer P., Pasquier J., "Web Services Technologies: State of Art Definitions, Standards, Case study", 2009, URL http://diuf.unifr.ch/drupal/softeng/sites/diuf.unifr.ch.drupal.softeng/files/file/publications/internal/WP09-04.pdf

Alrouh B., Ghinea G., "A Performance Evaluation of Security Mechanisms for Web services", International Conference on Information Assurance and Security, 18-20 August 2009, Xi'an, China, pp. 715-718. ISBN 978-0-7695-3744-3

Craig@AWS, "Amazon Web Services: Overview of security Process", Amazon, 2013, URL http://aws.amazon.com/articles/1697

Atadjanov A.J., "Exchanging bibliographic data with its full text by SOAP", 2010 4th International Conference on Application of Information and Communication Technologies (AICT), 12-14 October 2010, Tashkent, Uzbekistan, 2010, pp. 1-2, ISBN 978-1-4244-6903-1

Al-Zoubi K., Wainer G., "Using REST Web-Services Architecture for Distributed Simulation", ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation", 22-25 June 2009, Lake Placid, NY, USA, 2009, pp.114 – 121, ISBN 978-0-7695-3713-9

Artus D. J.N., "SOA realization: Service design principles", IBM white paper, 2006, URL http://www.ibm.com/developerworks/webservices/library/ws-soa-design/

Bartoletti M., Degano P., Ferrari G. L., Zunino R., "Semantics-Based Design for Secure Web Services", Software Engineering, IEEE Transactions, Volume 34, Issue 1, 2008, pp. 33-49, ISSN 0098-5589

BEA Systems, “Domain Model For SOA: Realizing the Business Benefit of Service-Oriented Architecture“, BEA white paper, 2005, URL http://www.soablueprint.com/yahoo_site_admin/assets/docs/BEA_SOA_Domains_WP.290214359.pdf

Booth D., Hass H., McCabe F., Newcomer E., Champion M., Ferris C., Orchard D., “Web Services Architecture”, W3C Working Group Note, 11 February 2004, URL http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

Boos G., Malladi P., Quan D., Legregni L., Hall H., “Cloud Computing”, IBM white paper, IBM Corporation, 2007

Bray T., Paoli J., Sperbery-McQueen C.M., Maler E., Yergeau F., Cowan J., “Extensible Markup Language (XML) 1.1 (Second Edition), W3C Recommendation, 2006, URL http://www.w3.org/TR/xml11/

Brisco T., “DNS Support for load Balancing”, RFC 1794, IETF, 1995, URL http://tools.ietf.org/html/rfc1794

Champion M., Ferris C., Newcomer E., Orchard D., “Web Services Architecture”, W3C Working Draft, 14 November 2002, URL http://www.w3.org/TR/2002/WD-ws-arch-20021114/

Change, L. C., Chiang H. K., Chiang W. Y., Smart-GIS: “A SVG-based tool for Visualizing and Monitoring of SARS Movement”, 3rd International Conference on Information Technology: Research and Education, 27-30 June 2005, pp. 282-286, ISBN 0-7803-8932-8

Chen S., Zic J., Tang K., Lavy D., “Performance Evaluation and Modeling of Web Services Security”, IEEE International Conference on Web Services, 9-13 July 2007, Salt Lake City, UT, USA, 2007, pp. 431-438, ISBN 0-7695-2924-0

Cheng F., Meinel C., “Design of Lock-Keeper Federated Authentication Gateway”, 11th International Conference on Advanced Communication Technology, 15-18 February 2009, Phoenix Park, Dublin, Ireland, 2009, pp. 1041-1046, ISBN 978-89-5519-138-7

Chonka A., Zhou W., Xiang Y., "Protecting Web Services with Service Oriented Traceback Architecture", 8th IEEE International Conference on Computer and Information Technology, 8-11 July 2008, Sydney, NSW, 2008, pp. 706-711, ISBN 978-1-4244-2357-6

Christensen E., Curbera F., Meredith G., Weerawarana S., "Web Services Description Language (WSDL) 1.1", W3C Note, 15 March 2001, URL http://www.w3.org/TR/wsdl

Damiani E, De Capitani di Vimercati S., Paraboschi S., Samarati P., "Secure SOAP E-Services", International Journal of Information Security (IJIS), Volume 1, Issue 2, 2002, pp. 100-115

Davis J., "Open Source SOA", Manning Publications Co., May 2009, ISBN 1933988541

DeMartini T., Nadalin A., Kaler C., Monzillo R., Hallam-Baker P., "Web Services Security Rights Expression Language (REL) Token Profile 1.1", OASIS Standard, 1 February 2006, URL http://docs.oasis-open.org/wss/v1.1/oasis-wss-rel-token-profile-1.1.pdf

Dierks T., Rescorla E., "The Transfer Layer Security (TLS) Protocol version 1.2", RFC5426, Network Working Group, 2008, URL http://www.ietf.org/rfc/rfc5246.-txt

Engelen R., Zhang W., "Identifying Opportunities for Web Services Security Performance Optimizations", IEEE Congress on Services – Part 1, 6-11 July 2008, Honolulu, HI, USA , 2008, pp. 209-210, ISBN 978-0-7695-3286-8

Engelen V., R.A., Zhang W., "An Overview and Evaluation of Web Services Security Performance Optimizations", IEEE International Conference on Web Services, 23-26 September 2008, Beijing, China, 2008, pp. 137-144, ISBN 978-0-7695-3310-0

Erl T. "Service-Oriented Architecture: Concepts, Technology, and Design", Prentice Hall/PearsonPTR, 2005, ISBN 0131858580

Erl T. "SOA Principles of Service Design", Prentice Hall/PearsonPTR, 2007, ISBN 0132344823

Fielding R. T., Irvine UC, Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., Berners-Lee T. "Hypertext Transfer Protocol HTTP/1.1", RFC2616, Network Working Group, 1999, URL http://www.w3.org/Protocols/rfc2616/rfc2616.html

Fielding R. T., "Architecture Styles and the Design of Network-based Software Architectures", University of California, Irvine, PhD. Dissertation, 2000, URL http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Foster I., Zhao Y., Raicu I., Lu S., "Cloud Computing and Grid Computing 360-Degree Compared", Grid Computing Environments Workshop, 12-16 November 2008, Austin, TX, USA, 2008, pp. 1-10, ISBN 978-1-4244-2860-1

Fusaro VA., Patil P., Gafni E., Wall D.P., Tonellato PJ., "Biomedical Cloud Computing with Amazon Web Services", PLOS Computational Biology, 2011, doi:10.1371/journal.pcbi.1002147, URI http://www.ploscompbiol.org/article/info%3Adoi%2F10.1371%2Fjournal.pcbi.1002147

Hansen M. D., "SOA Using Java Web Services", Prentice Hall Professional, 2007, ISBN 0130449687

Genge B., Haller P., "Extending WS-Security to Implement Security Protocols for Web Services", International Conference on Recent Achievements in Mechatronics, Automation, Computer-Sciences and Robotics, 20-21 March 2009, Tîrgu Mureş, 2009, Volume 1, pp.105-112, ISSN 2065-5916

Godfrey B., "A primer on distributed computing", 2006, URL http://www.bacchae.co.uk/docs/dist.html

Gottschalk K., "Web Services architecture overview", IBM Software Group, 2000, URL http://www.ibm.com/developerworks/webservices/library/w-ovr/

Gu Y. S., Zhang B. J., Xu W., "Research and Realization of Web Services Security Based on XML Signature", International Conference on Networking and Digital Society, 30-31 May 2009, Guiyang, Guizhou, China, 2009, pp. 116-118, ISBN 978-0-7695-3635-4

Gudgin M., Hadley M., Mendelsohn N., Moreau J. J., Nielsen H. F., Karmarkar A. Lafon Y., "SOAP Version 1.2 Part 1 : Messaging Framework (Second Edition)", W3C Recommendation, 27 April 2007, URL http://www.w3.org/TR/soap12-part1/

Gudgin M., Hadley M., Mendelsohn N., Moreau J. J., Nielsen H. F., Karmarkar A. Lafon Y., "SOAP Version 1.2 Part 2: Adjuncts (Second Edition)", W3C Recommendation, 27 April 2007, URL http://www.w3.org/TR/soap12-part2/

Gutierrez, C.; Fernandez-Medina, E.; Piattini, M., "PWSSec Process for Web Services Security", International Conference on Web Services, 18-22 September 2006, Chicago, IL, USA, 2006, pp. 213-222, ISBN 0-7695-2669-1

Gulbrabdsen A., Vixie P., Esibov L., "A DNS RR for specifying the location of services (DNS SRV)", RFC 2052, IETF, 2002, URL www.ietf.org/rfc/rfc2782.txt

Gao L., Liu S. F., Lu H., "A Solution of Axis2 Message Routing and Web Services Security", 6th International Conference on Pervasive Computing and Applications, 26-28 October 2011, Port Elizabeth, South Africa, 2011, pp.384-388, ISBN 978-1-4577-0209-9

Guo X., "A URL-Based System Model for Web Service Unified ID Authorization", International Conference on Image Analysis and Signal Proceedings, 11-12 April 2009, Taizhou, China, 2009, pp. 324-326, ISBN 978-1-4244-3987-4

Iacono L. L., Rajasekaran H., "Secure Browser-based Access to Web Services", IEEE International Conference on Communication, 14-18 June 2009, Dresden, Germany, 2009, pp. 1-5, ISBN 978-1-4244-3435-0

ITU-T, X.509, "Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks", INTERNATIONAL STANDARD ISO/IEC 9594-8, ITU-T Recommendation, 11/2008, URL http://www.itu.int/rec/T-REC-X.509-200811-I/en

Jia L., Zhang Z., "Research of Interoperability security between .net and J2EE", International Workshop on Intelligent Systems and Applications, 23-24 May 2009, Wuhan, China, pp. 1-3, ISBN 978-1-4244-3893-8

Judith M. Myerson, “Web Service Architecture”, Tect, Chicago, USA, 2009, URL http://www.onlinetechbooks.com/programming-books/WEBSERVICES_ARCHITECTURES.pdf

Juric, M.B., Zivkovic, A., Hericko, M., Brumen, B., Welzer, T., Rozman, I., “Performance assessment framework for distributed object architectures”, Advances in Databases and Information Systems, Springer Berlin Heidelberg, Lecture Notes in Computer Science Volume 1691, 1999, pp 349-366, ISBN 978-3-540-66485-7

Kosmajac D., “Information systems security and security extension in Jersey RESTful framework”, 20th Telecommunication Forum, 20-22 November 2012, Belgrade, Serbia, 2012, pp. 1556-1559, ISBN 978-1-4673-2983-5

Knap T., Mlynkova I., “Towards More Secure Web Services- Exploiting and analyzing XML signature security issues”, Third International Conference on Research Challenges in Information Science, 22-24 April 2009, Fez, 2009, pp. 49-58, ISBN 978-1-4244-2864-9

Kudo M., “PBAC: Provision-based access control model”, International Journal of Information Security, Springer-Verlag, February 2002, Volume 1, Issue 2, pp 116-130, ISSN 1615-5262

Kumari G. P., Kandan B., Mishra., “Experience sharing on SOA based Heterogeneous System Integration”, IEEE Congress on Services – Part I”, 6-11 July 2008, Honolulu, HI, USA, 2008, pp. 107-108, ISBN 978-0-7695-3286-8

Lavarack T., Coetzee M., “Considering web services Security policy compatibility”, Information Security for South Africa, 2-4 August 2010, Sandton, Johannesburg, 2010, pp. 1-8, ISBN 978-1-4244-5493-8

Lavarack T., Coetzee M., “Web services security policy assertion trade-offs”, Sixth International Conference on Availability, Reliability and Security (ARES), 22-26 August 2011, Vienna, 2011, pp. 535-540, ISBN 978-1-4577-0979-1

Lebanidze E., “Securing Enterprise Web Application at the Source: An Application Security Perspective”, The Open Web Application Security Project (OWASP), 2012, URL

https://www.owasp.org/images/8/83/Securing_Enterprise_Web_Applications_at_the_Source.pdf

Lee S. P., Chan L. P., Lee E. W., "Web Services Implementation Methodology for SOA Application", IEEE International Conference on Industrial Informatics", 16-18 August 2006, Singapore, 2006, pp. 335-340, ISBN 0-7803-9700-2

Lewis G. A., Morris E., Simanta S., Wrage L., "Common Misconceptions about Service-Oriented Architecture", 6th International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, 26 Feb – 2 March 2007, Banff, Alta, 2007, pp. 123-130, ISBN 0-7695-2785-X

Li J., Li B., Li L., Che T., "A Policy Language for Adaptive Web Services Security Framework", Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 30 July – 1 August 2007, Qingdao, China, 2007a, pp. 261-266, ISBN 978-0-7695-2909-7

Li J., Li B., Li L., Che T., "An Agent-based Policy Aware Framework for Web Services Security", IFIP International Conference on Network and Parallel Computing Workshops, 18-21 September 2007, Liaoning, China, 2007b, pp. 849-854, ISBN 978-0-7695-2943-1

Li Y., Peng Y. I., Zhan G. H., Zhang L., "The Research of Security Asynchronous Web Services based on SOA Architecture", IEEE International Conference on Networking, Sensing and Control, 6-8 April 2008, Sanya, China, 2008 pp. 1332-1336, ISBN 978-1-4244-1685-1

Liu Y., Yeap T. H., O'Brien W., "Securing XML Web Services with Elliptic Curve Cryptography", Canadian Conference on Electrical and Computer Engineering, 22-26 April 2007, Vancouver, BC, 2007, pp. 974-977, ISBN 1-4244-1020-7

Ma K., Song C. X., "Research on a Web Security Service System Structure Model", International Conference on Advanced Computer Theory and Engineering, 20-22 December 2008, Phuket, Thailand, 2008, pp. 884-887, ISBN 978-0-7695-3489-3

Maamar Z., Hacid H., Huhns M. N., "Why Web Services Need Social Network", IEEE Internet Computing, March – April 2011, Volume 15, Issue 2, pp. 90-94, ISSN 1089-7801

Mahmoud Q. H., "Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI)", Oracle Technology Network Article, April 2005, URL http://www.oracle.com/technetwork/articles/javase/soa-142870.html

Maler E. Mishra P., Philpott R., "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V1.1", OASIS Standard, 2 September 2003 URL https://www.oasis-open.org/committees/download.php/3406/

Meier J.D., Farre C., Taylor J., Bansode P., Gregersen S., Sundararajan M., Boucher R, "Improving Web Services Security-Scenarios and Implementation Guidance for WCF", WCF Security Guidance Project, Microsoft Corporation, February 2009, URL http://msdn.microsoft.com/en-us/library/ff650794.aspx

Meier J. D., Hill D., Homer A., Taylor J., Bansode P., Wall L., Boucher Jr. R., Bogawat, Lonnie A., "Microsoft Application Architecture Guide 2nd Edition", October 2009, URL http://msdn.microsoft.com/en-us/library/ff650706.aspx

Mitra N., Lafon Y., "SOAP Version 1.2 Part 0: Primer (Second Edition), W3C Recommendation, 27 April 2007, URL http://www.w3.org/TR/soap12-part0/

Mockapetris P., "Domain Names – Implementation and Specification", RFC 1035, IETF, 1987, URL www.ietf.org/rfc/rfc1035.txt

Monzillo R., Kaler C., Nadalin A., Hallem-Baker P., "Web Services Security SAML Token Profile 1.1", OASIS Standard, 1 February 2006, URL http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SAMLTokenProfile.pdf

Moralis A., Pouli V., Grammatikou M., Papavassiliou S., Maglaris V., "Performance Comparison of Web Services Security Kerberos Token Profile Against X.509 Token Profile", Third International Conference on Networking and Services", 19-25 June 2007, Athens, Greece, 2007, pp. 19-25, ISBN 978-0-7695-2858-9

Mourad A., Ayoubi S., Yahyaoui H., Otrok H., "New Approach for the Dynamic Enforcement of Web Services Security", Eighth Annual International Conference on Privacy Security and Trust, 17-19 August 2010, Ottawa, ON, Canada, 2010, pp. 189-196, ISBN 978-1-4244-7551-3

MSDN, "Web Services in Exchange 2013", Microsoft, 19 February 2013, URL http://msdn.microsoft.com/en-us/library/exchange/dd877012(v=exchg.150).aspx

Nadalin A., Kaler C., Monzillo R., Hallam-Baker P., "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", OASIS Standard Specification, 1 February 2006, 2006a, URL http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

Nadalin A., Kaler C., Monzillo R., Hallam-Baker P., "Web Services Security Username Token Profile 1.1", OASIS Standard Specification, 1 February 2006, 2006b, URL http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-UsernameTokenProfile.pdf

Nadalin A., Kaler C., Monzillo R., Hallam-Baker P., "Web Services Security X.509 Certificate Token Profile 1.1", OASIS Standard Specification, 1 February 2006, 2006c, URL http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-x509TokenProfile.pdf

Nadalin A., Kaler C., Monzillo R., Hallam-Baker P., "Web Services Security Kerberos Token Profile 1.1", OASIS Standard Specification, 1 February 2006, 2006d, URL http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-KerberosTokenProfile.pdf

Nakayama K., Ishizaki T., Oba M., "Application of Web Services Security Using Travel Industry Model", The 2005 Symposium on Applications and the Internet Workshops, 31-04 January 2005, pp. 358-361, ISBN 0-7695-2263-7

Neuman C., Yu T., Hartman S., Raeburn K., "The Kerberos Network Authentication Service (V5)", RFC 4120, IETF, 2005, URL www.ietf.org/rfc/rfc4120.txt

Nordbotten N. A., "XML and Web Services Security Standards", Journal of IEEE Communications Surveys & Tutorials, Publisher IEEE Communications, Volume 11 Issue 3, pp. 4-21, 2009, ISBN: 1553-877X

Nurse J.R.C., Sinclair J.E., “BOF4WSS A Business-Oriented Framework for Enhancing Web Services Security for e-Business”, Fourth International Conference on Internet and Web Applications and Services, 24-28 May 2009, Venice/Mestre , 2009, pp. 286-291, ISBN 978-1-4244-3851-8

Jendrock E., Ball J., Carson D., Evans I., Fordin S., Haase, K., “The Java EE 5 Tutorial For Sun Java System Application Server 9.1”, Oracle Corporation, September 2010, URL http://docs.oracle.com/javaee/5/tutorial/doc/

Oracle, “Java TM Platform, Standard Edition 6 API Specification”, Oracle, 2011, URL http://docs.oracle.com/javase/6/docs/api/

Parker D., “Toward a New Framework for Information Security”, The Computer Security Handbook (4th ed.), New York, John Wiley & Sons, Chapter 5, 2002, ISBN 0471412589

Peng D., Li C., Huo H., “An Extended UsernameToken-based Approach for REST-style Web Service”, 2nd IEEE International Conference on Computer Science and Information Technology, 8-11 August 2009, Beijing, China, 2009, pp. 582-586, ISBN 978-1-4244-4519-6

Phan T., Han J., Mueller I., Malinda K., “SOABSE- An Approach to Realizing Business-Oriented Security Requirements with Web Service Security Policies”, IEEE International Conference on Service-Oriented Computing and Applications, 14-15 January 2009, Taipei, 2009- pp. 1-10, ISBN 978-1-4244-5300-9

Postel J., “Internet Protocol“, RFC 791, IETF, 1981, 1981a, URL http://tools.ietf.org/html/rfc791

Postel J., “Transmission Control Protocol“, RFC 793, IETF, 1981, 1981b, URL http://tools.ietf.org/html/rfc793

Oh S., Park S., “Task-role-based access control model”, Journal of Information System, Elsevier Science Ltd, September 2003, Volume 28, Issue 6, pp. 533-562, ISSN 0306-4379

Omar S., “A message-level security approach for RESTful services”, Master Thesis, University of Oslo, 2011, URL https://www.duo.uio.no/handle/10852/8967

Qian Z., Zhang L., Yang J., Yang C., “Global SARS information WebGIS design and development”, International Geosciences and Remote Sensing Symposium, IGARSS’04, 20-24 September 2004, Volume 5, pp. 2861-2863, ISBN 0-7803-8742-2

Qu Z., Ge, Y., Jiang K., Lu T., “Key Issues in Building Web-based Services”, Third International Conference on Next Generation Web Services Practices, 29-31 October 2007, Seoul, Korea, 2007, pp. 119-122, ISBN 978-0-7695-3022-2

Quo C.F., Wu B., Wang M.D., “Development of a Laboratory Information System for Cancer Collaboration Projects”, 27th Annual Conference on Engineering in Medicine and Biology, Shanghai, China, 2007, ISBN 0-7803-8741-4

Rahaman M.A., Schaad A., “SOAP-based Secure Conversation and Collaboration”, IEEE International Conference on Web Services, 9-13 July 2007, Salt Lake City, USA, UT, 2007, pp. 471-480, ISBN 0-7695-2924-0

Rayns C., Clarke T., Conrad M., Wiese C., “SOAP Message Size Performance Considerations”, IBM redbooks, IBM International Technical Support Organization, 29 August 2007, URL http://www.redbooks.ibm.com/abstracts/redp4344.html

Rodriguez A., “RESTful Web Services: The basic”, IBM developerworks, 06 November 2008, URL http://www.ibm.com/developerworks/webservices/library/ws-restful/ws-restful-pdf.pdf

Sandhu R.S. , Coyne E.J. , Feinstein H.L., Youman C.E., “Role-based access control models”, Journal of Computer, IEEE Computer Society Press Los Alamitos, CA, USA, 2 February 1996, Volume 29, Issue 2, pp. 38-47

Sanchez-Villeda H., Schroeder S., Polacco M., McMullen M., Havermann S., Davis G., “Development of an integrated laboratory information management system for the maize mapping project,” Bioinformatics, US National Library of Medicine National Institutes of Health, 1 November 2003, Volume 19, Issue16, pp. 2022-2030, PMID:14594706

Serme G., Oliveria AS, Massiera J., Roudier Y., "Enabling Message Security for RESTful Services", 2012 IEEE 9th International Conference on Web Services (ICWS), 24-29 June 2012, Honolulu, HI, USA, 2012, pp. 114-121, ISBN 978-1-4673-2131-0

Schreier S., "Modeling RESTful applications", Second International Workshop on RESTful Design, March 2011, Hyderabad, India, ACM, 2011, pp. 15 -21, ISBN 978-1-4503-0623-2

Shah D., Patel D., "Dynamic and Ubiquitous Security Architecture for Global SOA", in Proceedings of the Second International Conference on Mobile Ubiquitous Computing, System, Services and Technologies, 29 September – 4 October 2008, Valencia, Spanish, 2008, pp. 482-487, ISBN 978-0-7695-3367-4

Shahgholi N., Mohsenzadeh M., Seyyedi M. A., Qorani S. H., "A new security framework against Web Services' XML attacks in SOA", 2011 7th International Conference on Next Generation Web Services Practices (NWeSP), 19-21 October 2011, Salamanca, northwestern Spain, 2011a, pp. 314-319, ISBN 978-1-4577-1125-1

Shahgholi N., Mohsenzadeh M., Seyyedi M. A., Qorani S. H., "A New SOA Security Framework Defending Web services Against WSDL Attacks", 2011 IEEE Third International Conference on Social Computing, 9-11 October 2011, Boston, MA, USA, 2011b, pp. 1259-1262, ISBN 978-1-4577-1931-8

Shaw M., Clements P., "A Field Guide to boxology: Preliminary Classification of Architectural Styles for Software Systems", The Twenty-First Annual International Computer Software and Applications Conference, (COMPSAC '97), 11-15 August 1997, Washington, DC, USA, 1997, pp. 6-13, ISBN 0-8186-8105-5

Sidharth N., Liu J., "IAPF A Framework for Enhancing Web Services Security", 31st Annual International Computer Software and Applications Conference, 24-27 July 2007, Beijing, China, 2007, pp. 23-30, ISBN 0-7695-2870-8

Singh S., Bawa S., "A Framework for Handling Security Problems in Grid Environment using Web Services Security Specification", International Conference on Semantics, Knowledge and Grid, November 2006, Guilin, Guangxi, China, 2006 pp. 68, ISBN 0-7695-2673-X

Sinha K.S., Sinha S., “Limitations of Web Service Security on SOAP Messages in a Document Production Workflow Environment”, 16th International Conference on Advanced Computing and Communications, 14-17 December 2008, Chennai, Indian , 2008, pp. 342-346, ISBN 978-1-4244-2962-2

Singhal A., “Web Services Security Challenges and Techniques”, Eighth IEEE International Workshop on Policies for Distributed System and Network, 13-15 June 2007, Bologna, Italian, 2007, pp. 282, ISBN 0-7695-2767-1

Singhal A., Winograd T., Scarfone K., “Guide to Secure Web Services”, Recommendations of the National Institute of Standards and Technology, U.S. Department of Commerce, NIST Special Publication 800-95, August 2007, URL http://csrc.nist.gov/publications/nistpubs/800-95/SP800-95.pdf

Sholler D., “2008 SOA User Survey: Adoption Trends and Characteristics”, Gartner, Inc., 26 September 2008, URL https://www.gartner.com/doc/765720

Tang K., Chen S., Levy D., Zic J., Yan B., “A Performance Evaluation of Web Services Security”, 10th IEEE International Enterprising Distributed Object Computing Conference, October 2006, Hong Kong, China, 2006, pp. 67-67, ISBN 0-7695-2558-X

TechTarget / Forrester Research, “State of SOA 2010”, TechTarget, Inc., June 2010, URL http://media.techtarget.com/searchSOA/downloads/TTAG-State-of-SOA-2010-execSummary-working-523%5B1%5D.pdf

Tekli J.M., “SOAP Processing Performance and Enhancement”, IEEE Transactions on Services Computing, Third Quarter, 24 February 2012, Volume 5, Issue 3, pp. 387-403, ISSN 1939-1374

Thurow K., Gode B., Dingerdissen U., Stoll N., “Laboratory information management systems for life sciences applications”, Organic Process Research and Development, American Chemical Society, 17 September 2004, Volume. 8, Issue 6, pp. 970-982,

Tsai W.T., Fan C., Chen Y., Paul R., Chung J. Y., “Architecture Classification for SOA-Based Application”, Ninth International Symposium on Object and Component-Oriented Real-Time Distributed Computing,24-26 April 2006, Gyeongju, China, 2006, pp. 295-302, ISBN 0-7695-2561-X

Wang J., Mao L., Cai., "A REST-based Approach to Integrate Enterprise Resources", International Forum on Computer Science-Technology and Applications", 25-27 December 2009, Chongqing, China, Volume 3, 2009, pp. 219-223, ISBN 978-0-7695-3930-0

World Health Organization, "International Classification of Diseases (ICD)", WHO, URL http://www.who.int/classifications/icd/en/

Wu J., Huang Z., "Proxy-based Web Service Security", IEEE Asia-Pacific Services Computing Conference, 9-12 December 2008, Yilan, Taiwan, 2008, pp. 1282-1288, ISBN 978-0-7695-3473-2

Xu P., Liu W. Y., "A Research of On-Line Static Security Analysis Based on WEB Services", 2011 Asia-Pacific Power and Energy Engineering Conference, 25-28 March 2011, Wuhan, China, 2011, pp. 1-4, ISBN 978-1-4244-6253-7

Yamaguchi Y., Chung H.V., Teraguchi M., Uramoto N., "Easy-To-Use Programming Model for Web Services Security", The 2nd IEEE Asia-Pacific Services Computing Conference, 11-14 December 2007, Tsukuba Science City, Japan, 2007, pp. 276-282, ISBN 0-7695-3051-6

Yamany H. F., Capretz M. A. M., "Use of Data Mining to Enhance Security for SOA", Third International Conference on Convergence and Hybrid Information Technology", 11-13 November 2008, Busan, South Korea, 2008, Volume 1, pp. 551-558, ISBN 978-0-7695-3407-7

Yu X., Bai Y., "A Method for Accessing Trusted Services Based on Service-Oriented Architecture", International Conference on Information Assurance and Security, 18-20 August 2009, Xian, China, 2009, Volume 2, pp. 685-688, ISBN 978-0-7695-3744-3

Yang Z., Liu Q., Zhao C., "A Context Based Dynamic Access Control Model for Web Service", IEEE/IFIP International Conference on Embedded and Ubiquitous Computing", 17-20 December 2008, Shanghai, China, 2008, pp. 339-343, ISBN 978-0-7695-3492-3

Zhang W., "Integrated Security Framework for secure Web Services", 2010 Third International Symposiums on Intelligent Information Technology and Security

Informatics", 2-4 April 2010, Jinggangshan, China, 2010, pp. 178-183, ISBN 978-1-4244-6730-3

Zhang Z., Wang K., Luan J., "A Combined Grid Security Approach Based on Web Services Security Specifications", ISECS International Colloquium on Computing, Communication, Control, and Management, 3-4 August 2008, Guangzhou, China, 2008, Volume 1, pp. 414-418, ISBN 978-0-7695-3290-5

Zhang J., "A Web Services-based Security Model for Digital Watermarking", 2011 International Conference on Multimedia Technology, 26-28 July 2011, Hangzhou, China, 2011, pp. 4805-4808, ISBN 978-1-61284-771-9

Zheng Y. H., "A Study on Network Security Technology Based on Web Service", 2011 International Conference on Computer Science and Service System, 27-29 June 2011, Nanjing , China, 2011, pp. 137-139, ISBN 978-1-4244-9762-1

Zhao F., Peng X., Zhao W., "Multi-Tier Security Feature Modeling for Service-Oriented Application Integration", Eighth IEEE/ACIS International Conference on Computer and Information Science, 1-3 June 2009, Shanghai, China, 2009, pp. 1178-1183, ISBN 978-0-7695-3641-5

APPENDIX A

– SOURCE CODE OF STUDENT WEB SERVICE AND TESTING CLIENT

Student Web Service is mainly composed of three java program files and one wsdl:

1. `StudnetService.java`
2. `StudentDao.java`
3. `Studnet.java`
4. `StudnetService.wsdl`

StudnetService.java

```
package uk.ac.sussex.webservices.method;

import uk.ac.sussex.dao.StudentDao;

import uk.ac.sussex.entity.Student;


public class StudentService {

	public Student getStudnet(long studentId){

		StudentDao studentDao=new StudentDao();

		Student student=studentDao.getStudentById(1);

		return student;

	}
```

```
}
```

StudnetDao.java

```
package uk.ac.sussex.dao;

import uk.ac.sussex.entity.Student;

public class StudentDao {

	public Student getStudentById(long studentId){

		Student student=new Student();

		student.setStudentId(studentId);

		student.setFirstName("Peter");

		student.setLastName("Gade");

		student.setGender("M");

		student.setNationality("Genermy");

		student.setPhone("12345678");

		student.setEmail("perter.gade@sussex.ac.uk");

		return student;

	}

}
```

Studnet.java

```
package uk.ac.sussex.entity;

import java.util.Date;

public class Student {

	private long studentId;

	private String firstName;

	private String lastName;

	private String gender;

	private String nationality;

	private Date dateOfBirth;

	private String phone;

	private String email;


	public long getStudentId(){

		return studentId;

	}

	public void setStudentId(long studentId){
```

```
        this.studentId=studentId;
}
public String getFirstName() {
        return firstName;
}
public void setFirstName(String firstName) {
        this.firstName = firstName;
}
public String getLastName() {
        return lastName;
}
public void setLastName(String lastName) {
        this.lastName = lastName;
}
public String getGender() {
        return gender;
}
public void setGender(String gender) {
```

```
        this.gender = gender;
}
public String getNationality() {
        return nationality;
}
public void setNationality(String nationality) {
        this.nationality = nationality;
}
public Date getDateOfBirth() {
        return dateOfBirth;
}
public void setDateOfBirth(Date dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
}
public String getPhone() {
        return phone;
}
public void setPhone(String phone) {
```

```
                this.phone = phone;

        }

        public String getEmail() {

                return email;

        }

        public void setEmail(String email) {

                this.email = email;

        }

}
```

StudnetService.wsdl

```
 <?xml version="1.0" encoding="UTF-8" ?>

- <wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:ns1="http://org.apache.axis2/xsd"
xmlns:ns="http://method.webservices.sussex.ac.uk" xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:ax21="http://entity.sussex.ac.uk/xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" targetNamespace="http://method.webservices.sussex.ac.uk">
```

```
  <wsdl:documentation>Please Type your service description here</wsdl:documentation>
- <wsdl:types>
- <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" targetNamespace="http://entity.sussex.ac.uk/xsd">
- <xs:complexType name="Student">
- <xs:sequence>
  <xs:element minOccurs="0" name="dateOfBirth" nillable="true" type="xs:date" />
  <xs:element minOccurs="0" name="email" nillable="true" type="xs:string" />
  <xs:element minOccurs="0" name="firstName" nillable="true" type="xs:string" />
  <xs:element minOccurs="0" name="gender" nillable="true" type="xs:string" />
  <xs:element minOccurs="0" name="lastName" nillable="true" type="xs:string" />
  <xs:element minOccurs="0" name="nationality" nillable="true" type="xs:string" />
  <xs:element minOccurs="0" name="phone" nillable="true" type="xs:string" />
  <xs:element minOccurs="0" name="studentId" type="xs:long" />
  </xs:sequence>
  </xs:complexType>
  </xs:schema>
- <xs:schema xmlns:ax22="http://entity.sussex.ac.uk/xsd" attributeFormDefault="qualified" elementFormDefault="qualified"
targetNamespace="http://method.webservices.sussex.ac.uk">
  <xs:import namespace="http://entity.sussex.ac.uk/xsd" />
```

```
- <xs:element name="getStudnet">
- <xs:complexType>
- <xs:sequence>
  <xs:element minOccurs="0" name="studentId" type="xs:long" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
- <xs:element name="getStudnetResponse">
- <xs:complexType>
- <xs:sequence>
  <xs:element minOccurs="0" name="return" nillable="true" type="ax21:Student" />
  </xs:sequence>
  </xs:complexType>
  </xs:element>
  </xs:schema>
  </wsdl:types>
- <wsdl:message name="getStudnetRequest">
  <wsdl:part name="parameters" element="ns:getStudnet" />
```

```
  </wsdl:message>
- <wsdl:message name="getStudnetResponse">
  <wsdl:part name="parameters" element="ns:getStudnetResponse" />
  </wsdl:message>
- <wsdl:portType name="StudentServicePortType">
- <wsdl:operation name="getStudnet">
  <wsdl:input message="ns:getStudnetRequest" wsaw:Action="urn:getStudnet" />
  <wsdl:output message="ns:getStudnetResponse" wsaw:Action="urn:getStudnetResponse" />
  </wsdl:operation>
  </wsdl:portType>
- <wsdl:binding name="StudentServiceSoap11Binding" type="ns:StudentServicePortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="getStudnet">
  <soap:operation soapAction="urn:getStudnet" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
```

```
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:binding name="StudentServiceSoap12Binding" type="ns:StudentServicePortType">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
- <wsdl:operation name="getStudnet">
  <soap12:operation soapAction="urn:getStudnet" style="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:binding name="StudentServiceHttpBinding" type="ns:StudentServicePortType">
  <http:binding verb="POST" />
```

```
- <wsdl:operation name="getStudnet">
  <http:operation location="getStudnet" />
- <wsdl:input>
  <mime:content type="text/xml" part="parameters" />
  </wsdl:input>
- <wsdl:output>
  <mime:content type="text/xml" part="parameters" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="StudentService">
- <wsdl:port name="StudentServiceHttpSoap11Endpoint" binding="ns:StudentServiceSoap11Binding">
  <soap:address location="http://localhost:8881/SussexWebServices/services/StudentService.StudentServiceHttpSoap11Endpoint/" />
  </wsdl:port>
- <wsdl:port name="StudentServiceHttpSoap12Endpoint" binding="ns:StudentServiceSoap12Binding">
  <soap12:address location="http://localhost:8881/SussexWebServices/services/StudentService.StudentServiceHttpSoap12Endpoint/" />
  </wsdl:port>
- <wsdl:port name="StudentServiceHttpEndpoint" binding="ns:StudentServiceHttpBinding">
```

```
<http:address location="http://localhost:8881/SussexWebServices/services/StudentService.StudentServiceHttpEndpoint/" />
</wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

Student Web Service Client is mainly composed of three java program files:

1. `TestClient.java`
2. `StudentServiceSub.java`
3. `StudentServiceCallbackHandler.java`

TestClient.java

```
package uk.ac.sussex.webservices.method;

import java.rmi.RemoteException;

import uk.ac.sussex.webservices.method.StudentServiceStub.GetStudnet;
```

```
import uk.ac.sussex.webservices.method.StudentServiceStub.GetStudnetResponse;

import uk.ac.sussex.webservices.method.StudentServiceStub.Student;

public class TestClient {

        public static void main (String[] args) throws RemoteException {

                StudentServiceStub stub =new StudentServiceStub();

                GetStudnet getStudent= new GetStudnet();

                getStudent.setStudentId(new Long(1));

                GetStudnetResponse response=stub.getStudnet(getStudent);

                Student student=response.get_return();

                System.out.println(student.getFirstName());

        }

}
```

StudentServiceStub.java (Part of)

```
public uk.ac.sussex.webservices.method.StudentServiceStub.GetStudnetResponse getStudnet(

uk.ac.sussex.webservices.method.StudentServiceStub.GetStudnet getStudnet0)
```

```
throws java.rmi.RemoteException

{
        org.apache.axis2.context.MessageContext _messageContext = null;
        try {
                org.apache.axis2.client.OperationClient _operationClient = _serviceClient
                                .createClient(_operations[0].getName());
                _operationClient.getOptions().setAction("urn:getStudnet");
                _operationClient.getOptions().setExceptionToBeThrownOnSOAPFault(
                                true);

                addPropertyToOperationClient(
                                _operationClient,
                                org.apache.axis2.description.WSDL2Constants.ATTR_WHTTP_QUERY_PARAMETER_SEPARATOR,
                                "&");

                // create a message context
```

```
_messageContext = new org.apache.axis2.context.MessageContext();


// create SOAP envelope with that payload
org.apache.axiom.soap.SOAPEnvelope env = null;


env = toEnvelope(getFactory(_operationClient.getOptions()
                .getSoapVersionURI()), getStudnet0,
                optimizeContent(new javax.xml.namespace.QName(
                                "http://method.webservices.sussex.ac.uk",
                                "getStudnet")), new javax.xml.namespace.QName(
                                "http://method.webservices.sussex.ac.uk",
                                "getStudnet"));


// adding SOAP soap_headers
_serviceClient.addHeadersToEnvelope(env);
// set the message context with that soap envelope
_messageContext.setEnvelope(env);
```

```
// add the message contxt to the operation client
_operationClient.addMessageContext(_messageContext);

// execute the operation client
_operationClient.execute(true);

org.apache.axis2.context.MessageContext _returnMessageContext = _operationClient
                .getMessageContext(org.apache.axis2.wsdl.WSDLConstants.MESSAGE_LABEL_IN_VALUE);
org.apache.axiom.soap.SOAPEnvelope _returnEnv = _returnMessageContext
                .getEnvelope();

java.lang.Object object = fromOM(
                _returnEnv.getBody().getFirstElement(),
                uk.ac.sussex.webservices.method.StudentServiceStub.GetStudnetResponse.class,
                getEnvelopeNamespaces(_returnEnv));

return (uk.ac.sussex.webservices.method.StudentServiceStub.GetStudnetResponse) object;
```

```
} catch (org.apache.axis2.AxisFault f) {

        org.apache.axiom.om.OMElement faultElt = f.getDetail();
        if (faultElt != null) {
                if (faultExceptionNameMap
                                .containsKey(new org.apache.axis2.client.FaultMapKey(
                                                faultElt.getQName(), "getStudnet"))) {
                        // make the fault by reflection
                        try {
                                java.lang.String exceptionClassName = (java.lang.String) faultExceptionClassNameMap
                                                .get(new org.apache.axis2.client.FaultMapKey(
                                                                faultElt.getQName(), "getStudnet"));
                                java.lang.Class exceptionClass = java.lang.Class
                                                .forName(exceptionClassName);
                                java.lang.Exception ex = (java.lang.Exception) exceptionClass
                                                .newInstance();
                                // message class
                                java.lang.String messageClassName = (java.lang.String) faultMessageMap
```

```
                        .get(new org.apache.axis2.client.FaultMapKey(
                                        faultElt.getQName(), "getStudnet"));
                java.lang.Class messageClass = java.lang.Class
                        .forName(messageClassName);
                java.lang.Object messageObject = fromOM(faultElt,
                        messageClass, null);
                java.lang.reflect.Method m = exceptionClass.getMethod(
                        "setFaultMessage",
                        new java.lang.Class[] { messageClass });
                m.invoke(ex, new java.lang.Object[] { messageObject });

                throw new java.rmi.RemoteException(ex.getMessage(), ex);
            } catch (java.lang.ClassCastException e) {
                // we cannot intantiate the class - throw the original
                // Axis fault
                throw f;
            } catch (java.lang.ClassNotFoundException e) {
                // we cannot intantiate the class - throw the original
```

```
            // Axis fault
            throw f;
        } catch (java.lang.NoSuchMethodException e) {
            // we cannot intantiate the class - throw the original
            // Axis fault
            throw f;
        } catch (java.lang.reflect.InvocationTargetException e) {
            // we cannot intantiate the class - throw the original
            // Axis fault
            throw f;
        } catch (java.lang.IllegalAccessException e) {
            // we cannot intantiate the class - throw the original
            // Axis fault
            throw f;
        } catch (java.lang.InstantiationException e) {
            // we cannot intantiate the class - throw the original
            // Axis fault
            throw f;
```

```
                        }

                } else {

                        throw f;

                }

        } else {

                throw f;

        }

} finally {

        if (_messageContext.getTransportOut() != null) {

                _messageContext.getTransportOut().getSender().cleanup(

                                _messageContext);

        }

}
```

StudentServiceCallbackHandler.java

```
/**

 * StudentServiceCallbackHandler.java

 *

 * This file was auto-generated from WSDL
```

```
* by the Apache Axis2 version: 1.6.1  Built on : Aug 31, 2011 (12:22:40 CEST)
*/
   package uk.ac.sussex.webservices.method;
   /**
    *  StudentServiceCallbackHandler Callback class, Users can extend this class and implement
    *  their own receiveResult and receiveError methods.
    */
   public abstract class StudentServiceCallbackHandler{
   protected Object clientData;

   /**
   * User can pass in any object that needs to be accessed once the NonBlocking
   * Web service call is finished and appropriate method of this CallBack is called.
   * @param clientData Object mechanism by which the user can pass in user data
   * that will be avilable at the time this callback is called.
   */
   public StudentServiceCallbackHandler(Object clientData){
       this.clientData = clientData;
```

```
}


/**
* Please use this constructor if you don't want to set any clientData
*/
public StudentServiceCallbackHandler(){
    this.clientData = null;
}


/**
 * Get the client data
 */
 public Object getClientData() {
    return clientData;
 }
       /**
        * auto generated Axis2 call back method for getStudnet method
        * override this method for handling normal response from getStudnet operation
```

```
     */
    public void receiveResultgetStudnet(
             uk.ac.sussex.webservices.method.StudentServiceStub.GetStudnetResponse result
                 ) {
    }
   /**
    * auto generated Axis2 Error handler
    * override this method for handling error response from getStudnet operation
    */
     public void receiveErrorgetStudnet(java.lang.Exception e) {
     }
}
```

APPENDIX B

– SOURCE CODE OF PDNT

PDNT is mainly composed of three java program files:

4. `ParticipantDomainNameTokenHandler.java`
5. `MessageUtil.java`
6. `DNSClient.java`

ParticipantDomainNameTokenHandler.java

```
package soapMessage;

import java.util.List;

import javax.xml.soap.MessageFactory;

import javax.xml.soap.MimeHeaders;

import javax.xml.soap.SOAPBody;

import javax.xml.soap.SOAPElement;

import javax.xml.soap.SOAPEnvelope;

import javax.xml.soap.SOAPHeader;

import javax.xml.soap.SOAPMessage;

import javax.xml.soap.SOAPPart;

import client.DNSClient;


public class ParticipantDomainNameTokenHandler {

        private SOAPMessage soapMessage;
```

```
private SOAPPart soapPart;
private SOAPEnvelope soapEnvelope;
private SOAPHeader soapHeader;
private SOAPBody soapBody;

public ParticipantDomainNameTokenHandler(SOAPMessage message){
        try{
                soapMessage=message;
                this.soapPart = soapMessage.getSOAPPart();
            this.soapEnvelope = soapPart.getEnvelope();
            this.soapHeader = soapEnvelope.getHeader();
            this.soapBody=soapEnvelope.getBody();
            if (soapHeader==null){
                soapHeader=MessageUtil.createSecureHeader(soapEnvelope);
            }
        }catch(Exception e){
                e.printStackTrace();
        }
}

public void addDomainNameToken(String domainName){
        try{
```

```
                soapEnvelope.addNamespaceDeclaration("pdn", MessageUtil.URI_PDN);
                SOAPElement security=(SOAPElement) soapHeader.getChildElements(soapHeader.createQName("Security", "wsse")).next();
                SOAPElement participantDominName=security.addChildElement("ParticipantDominName", "pdn");
                SOAPElement dn=participantDominName.addChildElement("DomainName", "pdn");
                dn.addTextNode(domainName);
        }catch (Exception e){
                e.printStackTrace();
        }
}

public String getDomainName(){
        SOAPElement participantDominName=(SOAPElement) soapHeader.getElementsByTagName("pdn:ParticipantDominName").item(0);
        SOAPElement dn=(SOAPElement) participantDominName.getElementsByTagName("pdn:DomainName").item(0);
        String domainName=dn.getTextContent();
        return domainName;
}

public String getLookupService(){
        SOAPElement participantDominName=(SOAPElement) soapHeader.getElementsByTagName("pdn:ParticipantDominName").item(0);
        SOAPElement dn=(SOAPElement) participantDominName.getElementsByTagName("pdn:DomainName").item(0);
        String domainName=dn.getTextContent();
        return domainName;
```

```
}

public boolean validate(){
        String lookupService=getLookupService();
        DNSClient.getAddressRecord(lookupService);
        return true;
}

public boolean validate(String senderIP){
        String lookupService=getLookupService();
        List <String> ipList=DNSClient.getAddressRecord(lookupService);
        for (int i=0;i<ipList.size();i++){
                if (ipList.get(i).equals(senderIP))
                        return true;
        }
        return false;
}

public SOAPMessage getSoapMessage() {
        return soapMessage;
}
```

```
public void setSoapMessage(SOAPMessage soapMessage) {
        this.soapMessage = soapMessage;
}

public SOAPPart getSoapPart() {
        return soapPart;
}

public void setSoapPart(SOAPPart soapPart) {
        this.soapPart = soapPart;
}

public SOAPEnvelope getSoapEnvelope() {
        return soapEnvelope;
}

public void setSoapEnvelope(SOAPEnvelope soapEnvelope) {
        this.soapEnvelope = soapEnvelope;
}

public SOAPHeader getSoapHeader() {
        return soapHeader;
```

```
}

public void setSoapHeader(SOAPHeader soapHeader) {
        this.soapHeader = soapHeader;
}

public SOAPBody getSoapBody() {
        return soapBody;
}

public void setSoapBody(SOAPBody soapBody) {
        this.soapBody = soapBody;
}

public static void main(String[] args) {
        try{
                String senderIP="";
                MessageFactory factory = MessageFactory.newInstance();
                MimeHeaders mimeHeaders = new MimeHeaders();
         mimeHeaders.addHeader("Content-Type","text/xml; charset=UTF-8");
                SOAPMessage soapMessage=factory.createMessage(mimeHeaders, MessageUtil.readFileToInputStream("soapMessage.xml"));
                ParticipantDomainNameTokenHandler tokenHandler=new ParticipantDomainNameTokenHandler(soapMessage);
```

```
                        tokenHandler.validate(senderIP);
                }catch (Exception e){
                        e.printStackTrace();
                }
        }
}
```

MessageUtil.java

```
package soapMessage;
import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.StringReader;
import java.io.StringWriter;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.SecureRandom;
import java.util.Iterator;
```

```
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.MimeHeaders;
import javax.xml.soap.Node;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.transform.OutputKeys;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.apache.xml.security.c14n.Canonicalizer;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;
```

```
public abstract class MessageUtil {
        public static final String URI_DS="http://www.w3.org/2000/09/xmldsig#";
        public static final String URI_WSSE="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd";
        public static final String URI_WSU="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd";
        public static final String URI_XENC="http://www.w3.org/2001/04/xmlenc#";
        public static final String URI_PDN="http://www.w3.org/2001/04/pdn#";

        public static void dumpDocument(Node root) throws TransformerException {
            Transformer transformer = TransformerFactory.newInstance().newTransformer();
            transformer.setOutputProperty(OutputKeys.METHOD, "xml");
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");
            transformer.transform(new DOMSource(root), new StreamResult(System.out));
            //transformer.transform(new DOMSource(root), sr);
            //return outText.toString();
          }

        public static void dumpDocument(org.w3c.dom.Node root) throws TransformerException {
//          StringWriter outText = new StringWriter();
//          StreamResult sr = new StreamResult(outText);
            Transformer transformer = TransformerFactory.newInstance().newTransformer();
            transformer.setOutputProperty(OutputKeys.METHOD, "xml");
            transformer.setOutputProperty(OutputKeys.INDENT, "yes");
```

```
    transformer.transform(new DOMSource(root), new StreamResult(System.out));
    //transformer.transform(new DOMSource(root), sr);
    //return outText.toString();
  }

public static void writeToFile(Node root,String fileName) {
        try{
                Transformer transformer = TransformerFactory.newInstance().newTransformer();
                File file=new File(fileName);
                transformer.transform(new DOMSource(root), new StreamResult(file));

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

public static InputStream readFileToFlat (String fileName){
        try {
                File xmlFile=new File(fileName);
                BufferedReader br = new BufferedReader(new FileReader(xmlFile));
                String line;
```

```
StringBuilder sb = new StringBuilder();
boolean addSpace=false;
while((line=br.readLine())!= null){
        if (line.length()>0){
//                String s=line.trim();
                String s=line.trim().replaceAll("\\r\\n", "");
                s.replaceAll("\\n", "");
                s=s.replaceAll("\\t", "");

                if (addSpace){
                        if (!s.substring(0,1).equals("<")){
                                sb.append(" ");
                                addSpace=false;
                        }
                }

                sb.append(s);
                if (!s.substring(s.length()-1).equals(">"))  {
                        addSpace=true;
                }
        }
}
```

```
                        InputStream is = new ByteArrayInputStream(sb.toString().getBytes("UTF-8"));
                        System.out.println ("file length="+sb.toString().length());
//                      System.out.println (sb.toString());
                        return is;
                }catch (Exception e){
                        e.printStackTrace();
                        return null;
                }
        }

        public static InputStream readFileToInputStream (String fileName){
                try {
                        FileInputStream fis = null;
                        File xmlFile=new File(fileName);
                        fis = new FileInputStream(xmlFile);
                        return fis;
                }catch (Exception e){
                        e.printStackTrace();
                        return null;
                }
        }
```

```
public static String readFileToString (String fileName){
        try {
                File xmlFile=new File(fileName);
                BufferedReader br = new BufferedReader(new FileReader(xmlFile));
                String line;
                StringBuilder sb = new StringBuilder();
                boolean addSpace=false;
                while((line=br.readLine())!= null){
                        if (line.length()>0){
                                String s=line.trim().replaceAll("\\r\\n", "");
                                s.replaceAll("\\n", "");
                                s=s.replaceAll("\\t", "");

                                if (addSpace){
                                        if (!s.substring(0,1).equals("<")){
                                                sb.append(" ");
                                                addSpace=false;
                                        }
                                }

                                sb.append(s);
                                if (!s.substring(s.length()-1).equals(">"))  {
```

```
                                        addSpace=true;
                                }
                        }
                }
                sb.append("\r\n\r\n");
                return sb.toString();
        }catch (Exception e){
                e.printStackTrace();
                return null;
        }
}
public static String serialize(Iterator elements){
        try {
                Transformer transformer = TransformerFactory.newInstance().newTransformer();
                StringWriter stw = new StringWriter();
                transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
                transformer.setOutputProperty(OutputKeys.STANDALONE,"yes");
                transformer.setOutputProperty(OutputKeys.MEDIA_TYPE,"text");
                while (elements.hasNext()) {
//                      Element element = (Element) elements.next();
                        Node element = (Node) elements.next();
                        if (element!=null)
```

```
                                transformer.transform(new DOMSource(element), new StreamResult(stw));
                        }

//                      return canonicalize(stw.toString());
                        String result=stw.toString().replace("xmlns:xsi=\"http://www.w3.org/1999/XMLSchema-instance\" ", "");
                        return result;
                }catch (Exception e) {
                        System.out.println("error in serialize");
                        e.printStackTrace();
                        return null;
                }

        }

        public static String serialize(Element element){
                try {
                        Transformer transformer = TransformerFactory.newInstance().newTransformer();
                        StringWriter stw = new StringWriter();
                        transformer.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
                        transformer.transform(new DOMSource(element), new StreamResult(stw));
//                      return canonicalize(stw.toString());
                        return stw.toString();
```

```
        }catch (Exception e) {
                e.printStackTrace();
                return null;
        }

    }

    public static String serialize(NodeList content) throws Exception { //XMLEncryptionException {
            org.apache.xml.security.Init.init();
        Canonicalizer canon = Canonicalizer.getInstance(Canonicalizer.ALGO_ID_C14N_WITH_COMMENTS);

            ByteArrayOutputStream baos = new ByteArrayOutputStream();
     canon.setWriter(baos);
     canon.notReset();
     for (int i = 0; i < content.getLength(); i++) {
         canon.canonicalizeSubtree(content.item(i));
     }
     baos.close();
     return baos.toString("UTF-8");
}

    public static String canonicalize(String xml){
```

```
        try{
                org.apache.xml.security.Init.init();
            Canonicalizer canon = Canonicalizer.getInstance(Canonicalizer.ALGO_ID_C14N_WITH_COMMENTS);
            byte canonXmlBytes[] = canon.canonicalize(xml.getBytes());
            return new String(canonXmlBytes);
        }catch (Exception e){
                e.printStackTrace();
                return null;
        }
}

public static String readFile(String fileName){
        try {
                File xmlFile=new File(fileName);
                BufferedReader br = new BufferedReader(new FileReader(xmlFile));
                String line;
                StringBuilder sb = new StringBuilder();
                boolean addSpace=false;
                while((line=br.readLine())!= null){
                        if (line.length()>0){
                                String s=line.trim();
                                s=s.replaceAll("\\t", "");
```

```
                        }
                }
                return sb.toString();
        }catch (Exception e){
                e.printStackTrace();
                return null;
        }

}

public static SOAPHeader createSecureHeader(SOAPEnvelope soapEnvelope) throws Exception{
        soapEnvelope.addNamespaceDeclaration("wsse", MessageUtil.URI_WSSE);
        SOAPHeader soapHeader=soapEnvelope.addHeader();
        soapHeader.addHeaderElement(soapEnvelope.createQName("Security","wsse"));
        return soapHeader;
}

public static KeyPair generateKeyPair(){
        try{
    //KeyPairGenerator kpg = KeyPairGenerator.getInstance("DSA");
    KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
```

```
        kpg.initialize(1024, new SecureRandom());
        KeyPair keypair = kpg.generateKeyPair();
        return keypair;
            } catch (Exception e){
                    e.printStackTrace();
                    return null;
            }
    }

    public static SecretKey GenerateSymmetricKey() throws Exception{
        String jceAlgorithmName = "AES";
        KeyGenerator keyGenerator = KeyGenerator.getInstance(jceAlgorithmName);
        keyGenerator.init(128);
        return keyGenerator.generateKey();
    }

    public static SecretKey GenerateKeyEncryptionKey() throws Exception {
        String jceAlgorithmName = "DESede";
        KeyGenerator keyGenerator = KeyGenerator.getInstance(jceAlgorithmName);
        SecretKey keyEncryptKey = keyGenerator.generateKey();
        return keyEncryptKey;
}
```

```
public static Document stringToDocument(String plainText) {
        try{
                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            InputSource is = new InputSource( new StringReader( plainText ) );
            Document d = builder.parse( is );
            return d;
        }catch (Exception e){
                e.printStackTrace();
                return null;
        }
}

public static SOAPMessage getSOAPMessage(String fileName){
        try {
                MessageFactory factory = MessageFactory.newInstance();
                MimeHeaders mimeHeaders = new MimeHeaders();
         mimeHeaders.addHeader("Content-Type","text/xml; charset=UTF-8");
                SOAPMessage soapMessage=factory.createMessage(mimeHeaders, MessageUtil.readFileToInputStream(fileName));
                return soapMessage;
```

```
                }catch (Exception e){
                        e.printStackTrace();
                        return null;
                }
        }

        public static byte[] inputStreamToByteArray(InputStream in) throws IOException{
                ByteArrayOutputStream baos = new ByteArrayOutputStream(1024);
                byte[] buffer = new byte[1024];
                int len;
                while((len = in.read(buffer)) >= 0){
                        baos.write(buffer, 0, len);
                }
                in.close();
                baos.close();
                return baos.toByteArray();
        }
}
```

DNSClient.java

```
package client;
import java.util.ArrayList;
import java.util.List;
```

```
import org.xbill.DNS.ARecord;
import org.xbill.DNS.Lookup;
import org.xbill.DNS.Record;
import org.xbill.DNS.SRVRecord;
import org.xbill.DNS.TXTRecord;
import org.xbill.DNS.Type;


public  class DNSClient {
        public static List<String> getSrvRecord(String lookupService){
                try {

                        List<String> hostList=new ArrayList<String>();
                        Record [] records = new Lookup(lookupService, Type.SRV).run();
                        for (int i = 0; i < records.length; i++) {
                                SRVRecord srvRecord=(SRVRecord) records[i];
                                hostList.add(srvRecord.getTarget().toString());
                        }
                        return hostList;
                }catch (Exception e){
                        e.printStackTrace();
                        return null;
```

```
        }
}

public static String getARecord(String hostName){
        try{
                Record [] records = new Lookup(hostName, Type.A).run();
                ARecord aRecord=(ARecord) records[0];
                return aRecord.getAddress().toString();

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

public static List<String> getAddressRecord(String lookupService){
        List<String> ipList=new ArrayList<String>();
        List<String> hostList=getSrvRecord(lookupService);
        for (int i=0;i<hostList.size();i++){
                ipList.add(getARecord(hostList.get(i)));
        }
```

```
        return ipList;
    }


    public static void main(String[] args) {
        try{
            long startMillis = System.currentTimeMillis();
            System.out.println(DNSClient.getAddressRecord("_pdn._tcp.sussex.ac.uk"));
            long endMillis = System.currentTimeMillis();
            Long processingTime=endMillis-startMillis;
            System.out.println("Query Time:" + processingTime + "ms");

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

APPENDIX C

– SOURCE CODE OF PERFORMANCE EVALUATION

The performance evaluation of PDNT is mainly composed of eight java program files:

1. HTTPServer.java
2. RequestProcessor.java
3. HTTPClient.java
4. SOAPDescruption.java
5. SOAPEncryption.java
6. SOAPSignature.java
7. UsernameToken.java
8. ParticipantDomainTokenHandler.java (shown in appendix A)

HTTPServer.java

```
package server;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;


public class HTTPServer {
    private ServerSocket serverSocket;
    private static final int HTTP_PORT = 8888;
```

```
    public void start(){
            start (HTTP_PORT);
    }

    public void start(int port){
            Socket socket;
            try{
                    serverSocket=new ServerSocket(port);
                    System.out.println("Server Started at port " + port);
                    while (true) {
                            socket=null;
//                          synchronized (serverSocket) {
                                    socket=serverSocket.accept();
//                          }
                            socket.setSoTimeout(0);
                            new RequestProcessor(socket).start();
                    }
            } catch (Exception e){
                    e.printStackTrace();
            }
    }

    public static void main(String[] args) {
            // TODO Auto-generated method stub
            HTTPServer httpServer=new HTTPServer();
```

```
            httpServer.start();
        }

    }
```

RequestProcessor.java

```
package server;

import java.io.BufferedReader;

import java.io.ByteArrayInputStream;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.io.PrintStream;

import java.net.Socket;


import javax.xml.soap.MessageFactory;

import javax.xml.soap.MimeHeaders;

import javax.xml.soap.SOAPMessage;


import org.apache.http.ProtocolVersion;

import org.apache.http.RequestLine;

import org.apache.http.message.BasicHttpEntityEnclosingRequest;

import org.apache.http.message.BasicRequestLine;


import soapMessage.ParticipantDomainNameTokenHandler;

import soapMessage.SOAPDecryption;

import soapMessage.SOAPSignature;

import soapMessage.UserNameToken;


public class RequestProcessor extends Thread{
```

```
    private Socket client;
    private InputStream input;
    private PrintStream output;
    private Long processingTime;
    private long startMillis;
    private long endMillis;

    public RequestProcessor(Socket socket){
            try{
                    startMillis = System.currentTimeMillis();
                    client=socket;
                    input =client.getInputStream();
                    output = new PrintStream(client.getOutputStream());
            }catch (Exception e){
                    e.printStackTrace();
            }
    }

    public void run(){
            try {
//                  System.out.println("Accepting HTTP Request");
                    parse(input);

                    sendResponse();
```

```
                output.close();
                input.close();

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

private void sendResponse(){
        String soapxml=buildSOAPResponse();
        output.println("HTTP/1.1 200 OK");
        output.println("Content-Type: text/xml; charset=utf-8");
        output.println("Content-length: "+ soapxml.length());
        output.println("");
        output.println(soapxml);
        output.flush();
}

private String buildSOAPResponse(){
        StringBuilder message= new StringBuilder();
        message.append("Processing Time:"+processingTime);
        return message.toString();
}

private RequestLine getRequestLine(String requestLine){
```

```
        String[] line=requestLine.split(" ");

        String method=line[0];
        String uri=line[1];
        String protocolLine=line[2];

        String protocol=protocolLine.substring(0,protocolLine.indexOf("/"));
        String version=protocolLine.substring(protocolLine.indexOf("/")+1);

        String[] versionDetail=version.split("\\.");
        int major=Integer.parseInt(versionDetail[0]);
        int minor=Integer.parseInt(versionDetail[1]);

        ProtocolVersion protocolVersion=new ProtocolVersion(protocol,major,minor);
        return new BasicRequestLine(method,uri,protocolVersion);

}

public void parse(InputStream input) {
        try {

                BufferedReader in = new BufferedReader(new InputStreamReader(input));
                char [] data=null;
```

```
        String line;
        line=in.readLine();

        BasicHttpEntityEnclosingRequest httpRequest=new BasicHttpEntityEnclosingRequest(getRequestLine(line));

        int contentLength=0;

        while (true) {
                line=in.readLine();
                String[] header=line.split(":");
                System.out.println(line);
                if (header.length>=2) {
                        httpRequest.addHeader(header[0].trim(), header[1].trim());
                        if (header[0].equals("Content-Length")){
                                contentLength=Integer.parseInt(header[1].trim());
                                data=new char[contentLength];
                        }
                }
                if (line.equals("")){
                        break;
                }
        }
}catch (Exception e){
        e.printStackTrace();
}
```

```
}

private void handleSoapMessage(String soapData){
        try {

                SOAPMessage soapMessage=getSOAPMesssage(soapData);
                handleSoapMessageUserNameToken(soapMessage);

                endMillis = System.currentTimeMillis();
                processingTime=endMillis-startMillis;
                System.out.println(processingTime);

        }catch (Exception e){
                System.out.println("error in handleSoapMessage");
        }
}

private SOAPMessage handleSoapMessageDecryption(SOAPMessage soapMessage){
        SOAPDecryption soapDecrption =new SOAPDecryption(soapMessage);
        soapDecrption.decrypt();
        soapDecrption.getSoapPart().getFirstChild();
        return soapDecrption.getSoapMessage();
}

private void handleSoapMessageValidation(SOAPMessage soapMessage){
```

```
        try{
                SOAPSignature soapSignature=new SOAPSignature();
                soapSignature.validation(soapMessage.getSOAPHeader());
        }catch(Exception e){
                e.printStackTrace();
        }
}

private void handleSoapMessagePDN(SOAPMessage soapMessage){
        try{
                ParticipantDomainNameTokenHandler tokenHandler=new ParticipantDomainNameTokenHandler(soapMessage);
                tokenHandler.validate();
        }catch(Exception e){
                e.printStackTrace();
        }
}

private void handleSoapMessageUserNameToken(SOAPMessage soapMessage){
        try{
                UserNameToken userNameToken=new UserNameToken(soapMessage);
                userNameToken.validatePasswordDigest();

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

```
    }


    private SOAPMessage getSOAPMesssage(String soapData){
            try{
                    MessageFactory factory = MessageFactory.newInstance();
                    MimeHeaders mimeHeaders = new MimeHeaders();
             mimeHeaders.addHeader("Content-Type","text/xml; charset=UTF-8");
                   SOAPMessage soapMessage=factory.createMessage(mimeHeaders, new ByteArrayInputStream(soapData.toString().getBytes("UTF-8")));

                    return soapMessage;
            }catch (Exception e){
                    System.out.println("error in getSOAPMessage");
                    return null;
            }
    }


}
```

HTTPClient.java

```
package client;

import java.io.BufferedReader;

import java.io.ByteArrayOutputStream;

import java.io.InputStream;

import java.io.InputStreamReader;


import org.apache.http.Header;

import org.apache.http.HttpEntity;

import org.apache.http.HttpResponse;

import org.apache.http.HttpVersion;

import org.apache.http.client.HttpClient;

import org.apache.http.client.methods.HttpGet;

import org.apache.http.client.methods.HttpPost;

import org.apache.http.entity.StringEntity;

import org.apache.http.impl.client.DefaultHttpClient;

import org.apache.http.params.BasicHttpParams;

import org.apache.http.params.HttpParams;

import org.apache.http.params.HttpProtocolParams;


import soapMessage.MessageUtil;


public class HTTPClient extends Thread {

    private String records="100";
```

```
private String fileName="soapMessagePDNUserName"+records+".xml";

HttpClient  httpclient=new DefaultHttpClient();

public HTTPClient(){
        httpclient.getParams().setParameter("http.socket.timeout", new Integer(0));
        httpclient.getParams().setParameter("http.connection.timeout", new Integer(0));
        httpclient.getParams().setParameter("http.connection-manager.timeout", new Integer(0));
        HttpProtocolParams.setUserAgent(httpclient.getParams(), "soapClient/1.1 searchTemplate/1.0");
}

public void run(){
        try{
                this.parsing(this.submit());
        }catch(Exception e){
                System.out.println("Exception in run");
        }
}

private HttpParams getHttpParams(){
        HttpParams params = new BasicHttpParams();
        HttpProtocolParams.setVersion(params, HttpVersion.HTTP_1_1);
        HttpProtocolParams.setContentCharset(params, "UTF-8");
        HttpProtocolParams.setUserAgent(params, "HttpComponents/1.1");
        HttpProtocolParams.setUseExpectContinue(params, true);
```

```
        return params;
}

public HttpResponse go(){
        try {
        HttpGet httpget = new HttpGet("http://www.pj.gov.mo");
        httpget.getParams().setParameter("http.socket.timeout", new Integer(0));
                return httpclient.execute(httpget);
        } catch (Exception e){
                e.printStackTrace();
                return null;
        }
}

public HttpResponse submit() throws Exception{
        StringEntity se=null;
        try {
                HttpPost httppost = new HttpPost("http://192.168.111.74:8888");
                httppost.getParams().setParameter("http.socket.timeout", new Integer(0));
                se = new StringEntity(buildSOAPRequest(),"UTF-8");
                se.setContentType("text/xml");
                httppost.setHeader("Content-Type","application/soap+xml;charset=UTF-8");
                httppost.setEntity(se);
                return httpclient.execute(httppost);
        }catch (Exception e){
```

```
                System.out.println("Exception in submit");
                return null;
        }
}

public void parsing(HttpResponse response){
        try {
                System.out.println("parsing");

                HttpEntity entity = response.getEntity();
                Header httpHeaders[]=response.getAllHeaders();

                for (int i=0;i<httpHeaders.length;i++){
                        Header header=httpHeaders[i];
                        System.out.print(header.getName()+ ":");
                        System.out.println(header.getValue());
                }

                if (entity != null) {
                        InputStream instream = entity.getContent();
                        BufferedReader in = new BufferedReader(new InputStreamReader(instream));
                        char[] str=new char[Integer.parseInt(response.getHeaders("Content-length")[0].getValue())];
                        in.read(str);
                        System.out.println(new String(str));
                }
```

```
		}catch (Exception e){
			System.out.println("Exception in parsing");
		}
	}

	private String buildSOAPRequest(){
		return MessageUtil.readFileToString(fileName);
	}



	public static void main(String[] args) {

		int noOfRequests=1;
		for (int i=0;i<noOfRequests;i++)
			{
				new HTTPClient().start();
				try {
					Thread.sleep(200);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
	}

}
```

SOAPDescription.java

```
package soapMessage;

import javax.xml.soap.MessageFactory;
import javax.xml.soap.MimeHeaders;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import com.sun.org.apache.xml.internal.security.utils.Base64;

public class SOAPDecryption {

    private SOAPMessage soapMessage;
    private SOAPPart soapPart;
    private SOAPEnvelope soapEnvelope;
    private SOAPHeader soapHeader;
```

```
private SOAPBody soapBody;
private byte[] symmetricKey;

public SOAPDecryption(SOAPMessage message ){
        try{

                this.soapMessage=message;
                this.soapPart = soapMessage.getSOAPPart();
            this.soapEnvelope = soapPart.getEnvelope();
            this.soapHeader = soapEnvelope.getHeader();
            this.soapBody=soapEnvelope.getBody();




        } catch (Exception e){
                System.out.println("exception in init");
                e.printStackTrace();
        }
}

public void decrypt(){
        try{
                SOAPElement cipherValue=(SOAPElement) soapHeader.getElementsByTagName("xenc:CipherValue").item(0);
                String keyEncryptedKey=cipherValue.getTextContent();
                symmetricKey=KeyManager.decryptSymmetricKey(Base64.decode(keyEncryptedKey));
```

```
            cipherValue=(SOAPElement) soapBody.getElementsByTagName("xenc:CipherValue").item(0);
            String cipherText=cipherValue.getTextContent();

            String plainText=KeyManager.doSymantecDecryption(symmetricKey, KeyManager.Algorithm_AES, Base64.decode(cipherText));

            //Merge Node
            Document doc=MessageUtil.stringToDocument(plainText);
            Node newChild=doc.getChildNodes().item(0);
            Node oldChild=soapBody.getChildNodes().item(0);
            soapBody.removeChild(oldChild);
            Node importNode=soapPart.importNode(newChild, true);
            soapBody.appendChild(importNode);
            Node securityNode=soapHeader.getElementsByTagName("wsse:Security").item(0);
            Node encryptedKeyNode=soapHeader.getElementsByTagName("xenc:EncryptedKey").item(0);
            securityNode.removeChild(encryptedKeyNode);

            soapEnvelope.removeNamespaceDeclaration("xenc");

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

}
```

```
public SOAPPart getSoapPart() {
        return soapPart;
}

public void setSoapPart(SOAPPart soapPart) {
        this.soapPart = soapPart;
}


public static void main(String[] args) {
        try {
                SOAPMessage soapMessage=MessageUtil.getSOAPMessage("soapMessageEncrypted100.xml");
                SOAPDecryption soapDecrption =new SOAPDecryption(soapMessage);

                long startMillis = System.currentTimeMillis();
                soapDecrption.decrypt();
                long endMillis = System.currentTimeMillis();


                MessageUtil.dumpDocument(soapDecrption.getSoapPart());
                Long processingTime=endMillis-startMillis;
        }catch (Exception e){
                e.printStackTrace();
        }
}
```

```
    public SOAPMessage getSoapMessage() {
            return soapMessage;
    }

    public void setSoapMessage(SOAPMessage soapMessage) {
            this.soapMessage = soapMessage;
    }
}
```

SOAPEncryption.java

```
package soapMessage;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.MimeHeaders;
import javax.xml.soap.Node;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;

import org.w3c.dom.Element;

import com.sun.org.apache.xml.internal.security.utils.Base64;

public class SOAPEncryption {
    private SOAPMessage soapMessage;
    private SOAPPart soapPart;
    private SOAPEnvelope soapEnvelope;
    private SOAPHeader soapHeader;
    private SOAPBody soapBody;
    private byte[] symmetricKey;
    private String encryptedSymmetricKey;
```

```
public SOAPEncryption(SOAPMessage soapMessage ){
        try{


                this.soapMessage=soapMessage;
                this.soapPart = soapMessage.getSOAPPart();
            this.soapEnvelope = soapPart.getEnvelope();
            this.soapHeader = soapEnvelope.getHeader();
            this.soapBody=soapEnvelope.getBody();

            if (soapHeader==null){
                soapHeader=MessageUtil.createSecureHeader(soapEnvelope);
            }

                symmetricKey=KeyManager.genSymmetricKey(KeyManager.Algorithm_AES);
                encryptedSymmetricKey=Base64.encode(KeyManager.encryptSymmetricKey(symmetricKey));

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

public void encrypt(String referenceName){
        try {
```

```
                addEncryptHeader(referenceName);
                String plainText=MessageUtil.serialize(soapBody.getChildElements());
                String cipherText=Base64.encode(KeyManager.doSymantecEncryption(symmetricKey, KeyManager.Algorithm_AES, plainText));

                while (soapBody.getChildElements().hasNext()){
                        soapBody.removeChild((Node) soapBody.getChildElements().next());
                }

                SOAPElement EncryptedData=soapBody.addChildElement("EncryptedData", "xenc");
                SOAPElement encryptionMethod=EncryptedData.addChildElement("EncryptionMethod", "xenc");
                encryptionMethod.setAttribute("Algorithm", "http://www.w3.org/2001/04/xmlenc#aes128-cbc");
                SOAPElement CipherData=EncryptedData.addChildElement("CipherData", "xenc");
                SOAPElement CipherValue=CipherData.addChildElement("CipherValue","xenc");
                CipherValue.addTextNode(cipherText);

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

}

private void addEncryptHeader(String referenceName){
        try{
                soapEnvelope.addNamespaceDeclaration("xenc", MessageUtil.URI_XENC);
                SOAPElement security=(SOAPElement) soapHeader.getChildElements(soapHeader.createQName("Security", "wsse")).next();
```

```
                SOAPElement encryptedKey=security.addChildElement("EncryptedKey", "xenc");
                SOAPElement encryptionMethod=encryptedKey.addChildElement("EncryptionMethod", "xenc");
                encryptionMethod.setAttribute("Algorithm", "http://www.w3.org/2001/04/xmlenc#rsa-1_5");
                SOAPElement keyInfo=encryptedKey.addChildElement("KeyInfo","ds","http://www.w3.org/2000/09/xmldsig#");
                SOAPElement securityTokenReference=keyInfo.addChildElement("SecurityTokenReference","wsse");
                SOAPElement x509IssuerSerial=securityTokenReference.addChildElement("X509IssuerSerial", "ds");
                SOAPElement X509IssuerName=x509IssuerSerial.addChildElement("X509IssuerName", "ds");
                X509IssuerName.addTextNode(KeyManager.getIssuerName());
                SOAPElement x509SerialNumber=x509IssuerSerial.addChildElement("X509SerialNumber", "ds");
                x509SerialNumber.addTextNode(KeyManager.getIssuerSerialNumber());

                SOAPElement cipherData=encryptedKey.addChildElement("CipherData", "xenc");
                SOAPElement cipherValue=cipherData.addChildElement("CipherValue", "xenc");
                cipherValue.addTextNode(encryptedSymmetricKey);

                SOAPElement referenceList=encryptedKey.addChildElement("ReferenceList", "xenc");
                SOAPElement dataReference=referenceList.addChildElement("DataReference" , "xenc");
                dataReference.setAttribute("URI", referenceName);
        }catch (Exception e){
                e.printStackTrace();
        }
}

public SOAPMessage getSoapMessage() {
        return soapMessage;
```

```
}

public SOAPPart getSoapPart() {
        return soapPart;
}

public SOAPEnvelope getSoapEnvelope() {
        return soapEnvelope;
}

public SOAPHeader getSoapHeader() {
        return soapHeader;
}

public SOAPBody getSoapBody() {
        return soapBody;
}

public static void main(String[] args) {
        try {
                String records="500";

                long startMillis = System.currentTimeMillis();

                MessageFactory factory = MessageFactory.newInstance();
```

```
            MimeHeaders mimeHeaders = new MimeHeaders();
        mimeHeaders.addHeader("Content-Type","text/xml; charset=UTF-8");
            SOAPMessage soapMessage=factory.createMessage(mimeHeaders, MessageUtil.readFileToInputStream("soapMessage.xml"));


            SOAPEncryption soapEncryption=new SOAPEncryption(soapMessage);
          soapEncryption.encrypt("#MsgBody");



          long endMillis = System.currentTimeMillis();
          MessageUtil.writeToFile(soapEncryption.getSoapPart(),"soapMessageEncrypted"+records+".xml");
          Long processingTime=endMillis-startMillis;
          MessageUtil.dumpDocument(soapEncryption.getSoapPart());
        }catch (Exception e){
            e.printStackTrace();
        }
}

public byte[] getSymmetricKey() {
        return symmetricKey;
}

public void setSymmetricKey(byte[] symmetricKey) {
        this.symmetricKey = symmetricKey;
}
```

```
    public String getEncryptedSymmetricKey() {
            return encryptedSymmetricKey;
    }

    public void setEncryptedSymmetricKey(String encryptedSymmetricKey) {
            this.encryptedSymmetricKey = encryptedSymmetricKey;
    }

}
```

SOAPSignature.java

```
package soapMessage;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import java.io.InputStream;

import java.math.BigInteger;

import java.security.KeyFactory;

import java.security.KeyPair;

import java.security.PrivateKey;

import java.security.PublicKey;

import java.security.spec.RSAPublicKeySpec;

import java.util.Collections;

import java.util.Iterator;


import javax.xml.crypto.dsig.CanonicalizationMethod;

import javax.xml.crypto.dsig.DigestMethod;

import javax.xml.crypto.dsig.Reference;

import javax.xml.crypto.dsig.SignatureMethod;

import javax.xml.crypto.dsig.SignedInfo;

import javax.xml.crypto.dsig.XMLSignature;

import javax.xml.crypto.dsig.XMLSignatureFactory;

import javax.xml.crypto.dsig.dom.DOMSignContext;

import javax.xml.crypto.dsig.dom.DOMValidateContext;

import javax.xml.crypto.dsig.keyinfo.KeyInfo;
```

```
import javax.xml.crypto.dsig.keyinfo.KeyInfoFactory;
import javax.xml.crypto.dsig.keyinfo.KeyValue;
import javax.xml.crypto.dsig.spec.C14NMethodParameterSpec;
import javax.xml.soap.MessageFactory;
import javax.xml.soap.MimeHeaders;
import javax.xml.soap.Node;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPEnvelope;
import javax.xml.soap.SOAPHeader;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPPart;

import org.w3c.dom.NodeList;
import org.w3c.dom.Text;

import com.sun.org.apache.xml.internal.security.utils.Base64;

public class SOAPSignature {

    public void sign(KeyPair keypair,String referenceName, SOAPHeader soapHeader){
            try {

                XMLSignatureFactory sigFactory = XMLSignatureFactory.getInstance();
                Reference ref = sigFactory.newReference(referenceName, sigFactory.newDigestMethod(DigestMethod.SHA1,
                    null));
```

```
                SignedInfo signedInfo = sigFactory.newSignedInfo(sigFactory.newCanonicalizationMethod(
                    CanonicalizationMethod.INCLUSIVE_WITH_COMMENTS, (C14NMethodParameterSpec) null), sigFactory
                    .newSignatureMethod(SignatureMethod.RSA_SHA1, null), Collections.singletonList(ref));
                KeyInfoFactory kif = sigFactory.getKeyInfoFactory();
                KeyValue kv = kif.newKeyValue(keypair.getPublic());
                KeyInfo keyInfo = kif.newKeyInfo(Collections.singletonList(kv));
                XMLSignature xmlSignature = sigFactory.newXMLSignature(signedInfo, keyInfo);

                PrivateKey privateKey = keypair.getPrivate();
                DOMSignContext sigContext = new DOMSignContext(privateKey, soapHeader.getFirstChild()); //include the signature element in
<wsse:Security>
                sigContext.putNamespacePrefix(XMLSignature.XMLNS, "ds");
                xmlSignature.sign(sigContext);
                    }catch (Exception e){
                            e.printStackTrace();
                    }
    }
    public boolean validation(SOAPHeader soapHeader){
        try{
            NodeList nodeList=soapHeader.getElementsByTagName("ds:Signature");
            DOMValidateContext valContext = new DOMValidateContext(getPublicKeyFromHeader(soapHeader), nodeList.item(0));
                    XMLSignatureFactory sigFactory = XMLSignatureFactory.getInstance();
                    XMLSignature xmlSignature = sigFactory.unmarshalXMLSignature(valContext);

                    boolean coreValidity=xmlSignature.validate(valContext);
```

```
                if (coreValidity == false) {
                    boolean sv = xmlSignature.getSignatureValue().validate(valContext);
                    System.out.println(Base64.encode(xmlSignature.getSignatureValue().getValue()));

                    if (sv == false) {
                        // Check the validation status of each Reference.
                        Iterator i = xmlSignature.getSignedInfo().getReferences().iterator();
                        for (int j=0; i.hasNext(); j++) {
                            boolean refValid = ((Reference) i.next()).validate(valContext);
                        }
                    }
                } else {
                }
                return coreValidity;
    }catch (Exception e){
        e.printStackTrace();
        return false;
    }
}

public PublicKey getPublicKeyFromHeader(SOAPHeader soapHeader){
         try {
               NodeList nodeList=soapHeader.getElementsByTagName("ds:Modulus");
               Node node=(Node) nodeList.item(0);
```

```
            Text text=(Text) node.getChildNodes().item(0);
            BigInteger mod=Base64.decodeBigIntegerFromText(text);

          nodeList=soapHeader.getElementsByTagName("ds:Exponent");
          node=(Node) nodeList.item(0);
          text=(Text) node.getChildNodes().item(0);
          BigInteger exp=Base64.decodeBigIntegerFromText(text);

          KeyFactory rsaFactory = KeyFactory.getInstance("RSA");
          RSAPublicKeySpec rsaKeyspec =new RSAPublicKeySpec(mod,exp);
          PublicKey publicKey = rsaFactory.generatePublic(rsaKeyspec);
          return publicKey;
        }catch (Exception e){
               e.printStackTrace();
               return null;
        }
  }


public static void main(String[] args) {

       try{
               long startMillis = System.currentTimeMillis();
               SOAPSignature soapSignature=new SOAPSignature();
               MessageFactory factory = MessageFactory.newInstance();
```

```
                MimeHeaders mimeHeaders = new MimeHeaders();

                 mimeHeaders.addHeader("Content-Type","text/xml; charset=UTF-8");

                SOAPMessage soapMessage=factory.createMessage(mimeHeaders,
                MessageUtil.readFileToInputStream("soapMessagePDNSigned100.xml"));

                SOAPPart soapPart = soapMessage.getSOAPPart();

            SOAPEnvelope soapEnvelope = soapPart.getEnvelope();

            SOAPHeader soapHeader = soapEnvelope.getHeader();

            if (soapHeader==null){

                soapHeader=MessageUtil.createSecureHeader(soapEnvelope);

            }




            SOAPBody soapBody = soapEnvelope.getBody();

            System.out.println(soapSignature.validation(soapHeader));


        } catch (Exception e){

                e.printStackTrace();

        }

    }

}
```

UsernameToken.java

```
package soapMessage;

import java.math.BigInteger;

import java.security.MessageDigest;

import java.security.SecureRandom;

import java.text.SimpleDateFormat;

import java.util.Date;


import javax.xml.soap.SOAPBody;

import javax.xml.soap.SOAPElement;

import javax.xml.soap.SOAPEnvelope;

import javax.xml.soap.SOAPHeader;

import javax.xml.soap.SOAPMessage;

import javax.xml.soap.SOAPPart;


import com.sun.org.apache.xml.internal.security.utils.Base64;


public class UserNameToken {

        private SOAPMessage soapMessage;

        private SOAPPart soapPart;

        private SOAPEnvelope soapEnvelope;

        private SOAPHeader soapHeader;

        private SOAPBody soapBody;
```

```
final static String xsdDateFormat="yyyy-MM-dd'T'HH:mm:ssZ";
private BigInteger nonce;
private String created;
private byte[] hash;

public UserNameToken(){

}

public UserNameToken(SOAPMessage message){
        try{
                soapMessage=message;
                this.soapPart = soapMessage.getSOAPPart();
            this.soapEnvelope = soapPart.getEnvelope();
            this.soapHeader = soapEnvelope.getHeader();
            this.soapBody=soapEnvelope.getBody();

            if (soapHeader==null){
                soapHeader=MessageUtil.createSecureHeader(soapEnvelope);
            }
        }catch(Exception e){
                e.printStackTrace();
        }

}
```

```
public boolean validatePasswordDigest(){
        SOAPElement userNameToken=(SOAPElement) soapHeader.getElementsByTagName("wsse:UsernameToken").item(0);
        SOAPElement passwordElement=(SOAPElement) userNameToken.getElementsByTagName("wsse:Password").item(0);
        String passwordDigest=passwordElement.getTextContent();
        SOAPElement nonceElement=(SOAPElement) userNameToken.getElementsByTagName("wsse:Nonce").item(0);
        String nonceBase64=nonceElement.getTextContent();
        SOAPElement createdElement=(SOAPElement) userNameToken.getElementsByTagName("wsu:Created").item(0);
        String created=createdElement.getTextContent();
        return validatePasswordDigest(nonceBase64, created, passwordDigest);
}

private BigInteger generateNonce(){
        try{
                SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
                String randomValue=new Integer(random.nextInt()).toString();
                return new BigInteger(randomValue);
        }catch(Exception e){
                e.printStackTrace();
                return null;
        }
}

private String getDate(){
        return new SimpleDateFormat(xsdDateFormat).format(new Date());
```

```
}

private Date dateParse(String date){
	try{
		return new SimpleDateFormat(xsdDateFormat).parse(date);
	}catch(Exception e){
		e.printStackTrace();
		return null;
	}
}

private byte[] getSha1Hash(String text){
	try{
		MessageDigest sha = null;
		sha = MessageDigest.getInstance("SHA-1");
		sha.update(text.getBytes());
		return sha.digest();

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

public String createPasswordDigest(String password){
```

```
        nonce=generateNonce();
        created=getDate();
        hash=getSha1Hash(nonce+created+password);
        return Base64.encode(hash);
}

public String createPasswordDigest(BigInteger nonce,String created,String password){
        hash=getSha1Hash(nonce+created+password);
        return Base64.encode(hash);
}


public BigInteger getNonce() {
        return nonce;
}

public String getNonceBase64(){
        return Base64.encode(nonce);
}

public void setNonce(BigInteger nonce) {
        this.nonce = nonce;
}

public void setNonce(byte[] nonce){
```

```
        this.nonce=new BigInteger(nonce);
}

public String getCreated() {
        return created;
}

public void setCreated(String created) {
        this.created = created;
}

public boolean validatePasswordDigest(String nonceBase64,String created,String passwordDigest){
        try{
                BigInteger n=new BigInteger(Base64.decode(nonceBase64));
                String digest=createPasswordDigest(n,created,"Test");
                if (digest.equals(passwordDigest))
                        return true;
                else
                        return false;

        }catch(Exception e){
                e.printStackTrace();
                return false;
        }
}
```

```
	public static void main(String[] args)throws Exception{
		UserNameToken un=new UserNameToken();
		System.out.println(un.validatePasswordDigest("zgniDw==", "2012-02-10T16:45:27+0800", "NtaIDHV2y+beT9ED5IUUck9dvqE="));

	}
}
```