DEPLOYING AND INVOKING SECURE WEB SERVICES OVER JXTA FRAMEWORK

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

İLHAMİ GÖRGÜN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

DECEMBER 2004

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Ayşe Kiper
Chair of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Asuman Doğaç
Supervisor

Examining Committee Members

Assoc. Prof. Dr. Nihan K. Çiçekli	(METU, CENG)	
Prof. Dr. Asuman Doğaç	(METU, CENG)	
Assoc. Prof. Dr. İ. Hakkı Toroslu	(METU, CENG)	
Assoc. Prof. Dr. Ali Doğru	(METU, CENG)	
Bülent Kunaç	(Tepe Tech.)	

ABSTRACT

DEPLOYING AND INVOKING SECURE WEB SERVICES OVER JXTA FRAMEWORK

Görgün, İlhami
M.Sc., Department of Computer Engineering
Supervisor : Prof. Dr. Asuman Doğaç

December 2004, 113 pages

Web services introduce a new paradigm for distributed computing, and the technology that it introduces constructs a new type of Web application. Web services can be described as any software that makes its discovery and invocation available over the Internet, and uses a standardized XML messaging system.

The term peer-to-peer refers to a class of decentralized systems enabling the access of shared resources available on peers that are acting both as client and as server.

In this work, a peer-to-peer approach is used to expoit Web service technologies by providing Web service security for JXTA peer-to-peer networks. JXTA is a network programming environment that has particularly been designed for the peer-to-peer platform.

In order to achieve the goal of secure Web services, the specifications “WS-Security”, “XML Key Management Specification”, “WS-Trust” and “WS-SecurityPolicy” are exploited. “WS-Security” is primarily a specification for an XML-based security metadata container, and is a building block for the specifications “WS-Trust” and “WS-SecurityPolicy”. “WS-Trust” defines the process of how to acquire security tokens. Within the peer-to-peer network that is proposed with this work, a peer is dedicated to act as a “trusted third party” and to manage the processes for incorporating the security of public-key infrastructure, which is defined by “XML Key Management Specification”. In addition, the same peer is dedicated to manage to acquire security tokens, which is defined by “WS-Trust”. As for “WS-SecurityPolicy”, Web service invoking peers conform to this specification that specifies how to define security assertions stating Web service provider’s preferences and requirements.

This work realizes and achieves the necessity of bringing together the technologies mentioned above in order to propose an architecture of secure SOAP messaging for Web service invocation in peer-to-peer environment that is provided by the JXTA framework.

The work presented in this thesis is realized as a part of the SATINE project funded by the European Commission.

Keywords: Security, Web service, Peer-to-peer, JXTA

ÖZ

JXTA ÇATISI İÇİN GÜVENLİ AĞ SERVİSLERİ YAYILMASI VE KULLANILMASI

Görgün, İlhami
Y. Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Asuman Doğaç

Aralık 2004, 113 sayfa

Ağ servisleri dağıtık çalışma için yeni bir görüş tanıtmaktadır ve tanıttıkları teknoloji yeni ağ uygulama tekniği inşa etmektedir. Ağ servisleri, Internet üzerinden bulunmalarını ve kullanılmalarını mümkün kılan herhangi bir yazılım parçası olarak tanımlanabilir ve standartlaşmış XML mesajlaşma sistemini kullanmaktadır.

Eşler-arası terimi, hem alıcı hem de sunucu olarak davranan eşlerin paylaşılmış kaynaklarına ulaşımı mümkün kılan merkezileştirilmemiş sistemler sınıfını tanımlamaktadır.

Bu çalışmada, JXTA eşler-arası ağları için ağ servisi güvenliği sağlayarak ağ servisi teknolojilerinden faydalanabilmek için eşler-arası yaklaşımı kullanılmaktadır. JXTA, özellikle eşler-arası platformu için tasarlanmış bir ağ programlama ortamıdır.

Ağ servisleri için güvenliği sağlayabilmek için “WS-Security”, “XML Key Management Specification”, “WS-Trust” ve “WS-SecurityPolicy” tanımlamaları kullanılmaktadır. “WS-Security” temelde XML tabanlı yardımcı güvenlik veri içeriği için tanımlamadır ve “WS-Trust” ve “WS-SecurityPolicy” tanımlamaları için temel taş oluşturmaktadır. “WS-Trust” güvenlik jetonlarına ulaşma sürecini belirtmektedir. Bu çalışma ile tanıtılan eşler-arası ağ içerisinde bir eş, “güvenilir üçüncü sistem” olarak davranmak ve “XML Key Management Tanımlaması” ile belirtilen public-key mekanizmasının güvenliği ile ilgili süreçleri yerine getirmek işlemlerine adanmış durumdadır. Buna ek olarak, aynı eş, güvenlik jetonlarına ulaşma işlemini yerine getirmektedir. “WS-SecurityPolicy” tanımlamasına gelince ise, ağ servis kullanan eşler, ağ servis sağlayan eşlerin tercihlerini ve gereksinimlerini belirten bu tanımlamaya uymaktadırlar.

Bu çalışma, JXTA çatısının imkan verdiği eşler-arası ortamında ağ servislerinin kullanımı için, güvenli SOAP mesajlaşma mimarisi tanıtma amacı için yukarıda bahsedilen teknolojileri biraraya getirmektedir.

Bu çalışma, Avrupa Komisyonu tarafından desteklenen SATINE projesinin bir parçası olarak gerçekleştirilmiştir.

Anahtar Kelimeler: Güvenlik, Ağ servisi, Eşler-arası, JXTA

To My Family

ACKNOWLEDGMENTS

I would like to thank my supervisor Asuman Doğaç for all her guidance, advice, encouragement and support during this study.

I would like to thank each member of SRDC team for their technical guidance and support during this study.

I would like to thank Yıldıray Kabak, Gökçe Banu Laleci and Ümit Lütfü Altıntakan for their technical assistance and support throughout this study.

Finally, I would like to thank my family, especially to my fiancee and to my mother, for all their patience and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Peer-to-peer systems introduce a class of decentralized systems enabling the access of shared resources available on peers that are acting both as client and as server. The shared resources include computing power, data (storage and content), and network bandwidth. These features of peer-to-peer computing facilitate distributed computing, data/content sharing, communication and collaboration, or platform services.

Peer-to-peer computing is increasingly receiving attention in research, product development and investment. Peer-to-peer systems provide the advantages of improved scalability and reliability, resource aggregation and interoperability, increased autonomy, and dynamism.

Web services are autonomous platform-independent computational elements that can be described, published, discovered, orchestrated, and programmed using XML artefacts for the purpose of developing distributed interoperable applications. Well-accepted standards like Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP) make it possible to dynamically invoke Web services.

Despite the promise, Web services have lacked the mechanism for implementing security, which is a crucial aspect to enable Web services techbology to become more popular and common. In order to overcome this issue, specifications like WS-Security, WS-Trust, WS-Policy, and WS-

SecurityPolicy have emerged. The aim of this work is to bring together the technologies introduced by peer-to-peer computing and Web services by providing a secure Web service invoking mechanism in a peer-to-peer network. In this work, a peer-to-peer network architecture which provides secure Web service implementation conforming to the standards WS-Security, WS-Trust, WS-SecurityPolicy and XML Key Management Specification is implemented. The overall system architecture is denoted in Figure 1.1.

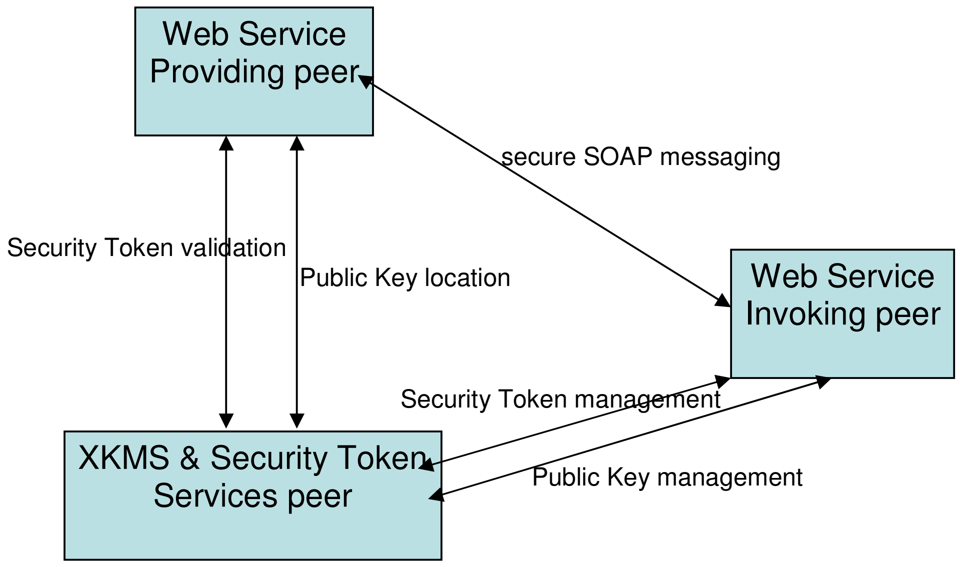

Web Service Providing peer
secure SOAP messaging
Security Token validation
Public Key location
Web Service Invoking peer
Security Token management
XKMS & Security Token Services peer
Public Key management

Figure 1.1. An Overall Architecture of the System

Within this work, in order to achieve peer-to-peer network creation and implementation, JXTA technology is exploited. JXTA technology is a network programming and computing platform that is designed to solve a number of problems in modern distributed computing, particularly in the area of peer-to-peer computing. The JXTA project has been initiated by Sun Microsystems and has been designed with the participation of a number of experts from academic institutions and industry.

The focus of this thesis is on defining an architecture to provide secure Web services for peer-to-peer systems, which has gained an increasing popularity. In this work, JXTA framework is exploited for the peer-to-peer network implementation. JXTA framework has introduced its own security mechanism [22], which provides whole message integrity and confidentiality only among the communicating peer pairs, and exploits the technology Transport Layer Security (TLS). For the Web services technology, a more complex mechanism is required in order to involve all the peers (Web service providing and invoking peers) within the peer-to-peer network by conforming to the well-accepted specifications introduced for Web services security.

The thesis work contributes to the SATINE project [24, 25, 26, 27, 28], which is developed within the Sixth Framework Programme of European Commission. The objective of the project is to develop a secure semantic-based interoperability framework for exploiting Web service platforms in conjunction with peer-to-peer networks in the travel industry.

The thesis is organized as follows: Chapter II describes the technologies that enable this work, namely, peer-to-peer computing, JXTA Framework, Web Services, Web Services Security, XML Key Management Specification, Web Services Trust, and Web Services Security Policy. Chapter III describes how the mechanism for performing secure Web services in peer-to-peer systems is achieved. Chapter IV describes the building blocks for the implementation of the introduced system, and presents a sample walkthrough for demonstrational purpose. Chapter V describes the integration of the thesis work to the SATINE project. Chapter VI summerizes the related work. Chapter VII concludes the thesis and presents the future work.

CHAPTER II

ENABLING TECHNOLOGIES

In this chapter, the technologies that enable this work are described. Initially, peer-to-peer computing, and JXTA technology that provides the framework to implement peer-to-peer computing are introduced in Section 2.1 and Section 2.2. Later in the section, Web services and the standards related with Web servives that are exploited in this work are introduced.

2.1. Peer-to-peer Computing

Peer-to-Peer (P2P) computing refers to the exchange of data between two computers. The P2P architecture allows to develop a decentralized application design where each computer, independent of software and hardware platforms, can access shared resources available on other peers.

In a typical client/server computing model, the communication between the computers on a network is through a centralized server. The server controls the flow of data and information between the client computers. Alternatively, P2P allows to share resources independently, without going through a central server. In P2P computing, numerous computers are interconnected with each other to share resources directly; and in a P2P application, multiple peers perform specific roles in communicating with each other.

The P2P computing model has several advantages over the centralized model. These advantages include effective utilization of network resources, dynamic nature of the network, and security and integrity of data. The distributed architecture offers interoperability among network devices. P2P is based on the concept of an equal participation of peers allowing resource sharing and collaboration among the integral parts of a distributed system design.

The following figure denotes the client/server model where client computers are connected with the Internet and the Web Server through HTTP:

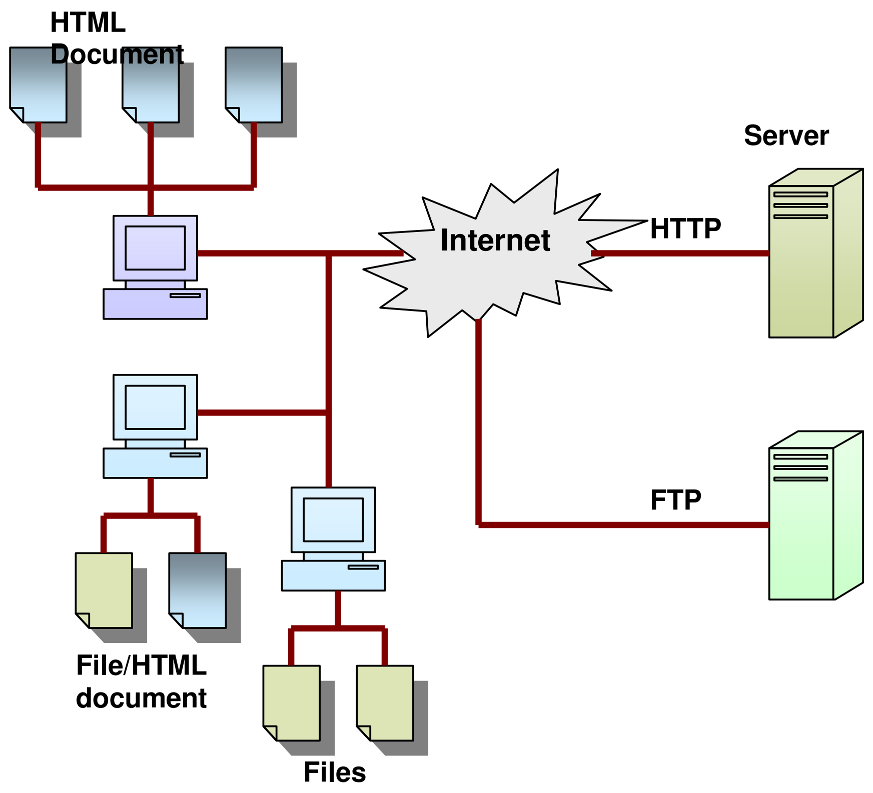


Figure 2.1. The Client/Server Model

In P2P computing model, the participating computers can work as clients as well as servers. A peer computer acts as a client but has an additional layer of software that allows it to act as a server. For example, a peer computer responds to the requests of other peers. These requests could be for retrieval or storage of information, a computation to be performed, or simply an exchange of messages. In this way, a peer can directly interact with other computers on the network without the need of a central server. The following figure denotes a sample P2P network:

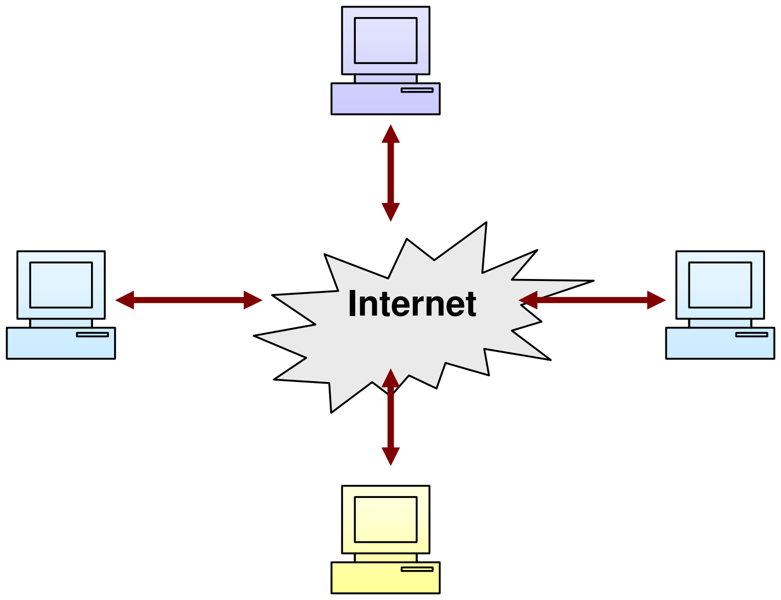


Figure 2.2. A P2P Network

In a P2P network, collaboration between the peer computers and resource sharing help in increasing productivity and reducing costs through efficient use of the available resources.

The SETI@Home project is a good example of how P2P applications can leverage unused resources and gain access to vast storage and computational power. This project comprises about 2.4 million users from over 200 countries who contribute the idle cycles on their computers to process data for Search for Extra Terrestrial Intelligence (SETI). Napster introduced another concept of distributed file sharing and file storage. In the Napster model, individual computer nodes connect with each other to copy MP3 files. Replication of content and information is useful because the client does not have to depend on a centralized server for the required data. Every node or peer can be used as a repeater to retransmit data once it reaches the node. In this way, Napster users can have access to several terabytes of storage and bandwidth at no extra cost.

With new advances in P2P technologies, many new concepts have been introduced. Enterprises are considering P2P as a viable option for cost reduction and efficient resource management. Companies, such as SUN and Groove networks, have taken up many innovative projects. With XML becoming the de facto standard for Web data, enterprises would be able to use Web Services in P2P applications for performing several business-to-business (B2B) related tasks. For example, buyers can use P2P networks to search for and find supplier and product details. This would help in direct B2B transactions and eliminate the need for intermediaries. SUN has taken initiative in promoting P2P by introducing JXTA, a collection of low-level protocols for developing P2P applications. According to SUN, JXTA can facilitate interoperability between a range of devices, such as personal computers, cell phones and other wireless devices.

The P2P model is significant in enterprises because it promises the following benefits:

- Overall cost of processing can be brought down and productivity can be increased.
- Value added services could be offered in business-to-business (B2B) and business-to-customer (B2C) transactions.

In the last years, there has been a rapid increase in the development of new P2P technologies and standards, which are particularly targeting interoperability. The most significant way to achieve interoperability is through a common infrastructure. A common platform reduces the efforts needed to develop applications for different platforms and interfaces. Project JXTA is a set of low-level protocol that allows programmers to develop efficient P2P applications. New generations of wireless devices embedded with JXTA are to deliver a significant level of interoperability. Figure 2.3. denotes the aimed P2P framework where applications and devices are interoperable:

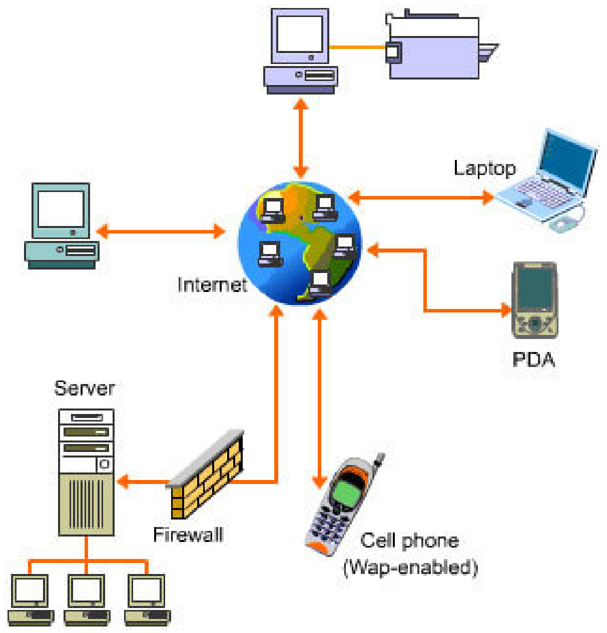


Figure 2.3. Target Architecture of P2P Framework

Finally, it is worth noting that issues such as security and trust need to be addressed in order to make the best use of P2P applications and to develop a standard platform for application developers, which constitute the target of this work, particularly for the exploitation of Web service technology within P2P network.

2.2. JXTA Framework

The project JXTA introduces a set of protocols, which enables peer-to-peer communication among computers and software components. JXTA is a network programming and computing platform that is designed to solve a number of problems in modern distributed computing, particularly in the area of peer-to-peer computing.

Project JXTA has originally been initiated by Sun Microsystems and designed with the participation of a number of experts from academic institutions and industry. The project has defined the following objectives based on the advantages of peer-to-peer systems:

- **Interoperability:** compatible P2P systems and participation among P2P systems.
- **Platform Independence:** independent of programming languages, development environments, or deployment platforms.
- **Ubiquity:** implementable on every device with a digital heartbeat, including sensors, consumer electronics, PDA's, network routers, desktop computers, and storage systems.

JXTA defines a set of protocols that enable computers connected in parallel to communicate with each other. Computers connected in parallel contain entities, called peers. That is, a peer is an entity that can speak the protocols required of a peer.

In a peer-to-peer model, each peer or entity in a parallel network has an ID. User-defined entities or a peer group represent a collection of peers. A unique ID also identifies a peer group. A peer can belong to multiple peer groups and can discover other entities dynamically. JXTA supports three types of communication in a peer-to-peer model, unicast, secure, and broadcast pipe. The following figure denotes the peer-to-peer model using the JXTA protocol:

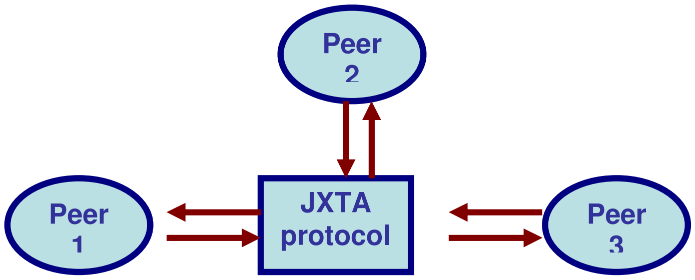


Figure 2.4. The Peer-to-Peer Model

JXTA defines peer group as a virtual entity that speaks the set of peer group protocols. Typically, a peer group is a collection of cooperating peers providing a common set of services. There is a special group, called the World Peer Group, which includes all JXTA peers.

JXTA exploits pipes as communication channels for sending and receiving messages. Pipes are asynchronous and unidirectional. Thus, there are input and output pipes between the peers. Pipes are also virtual, that is, an endpoint of a pipe can be bound to one or more peer endpoints.

JXTA has defined the following six protocols that provide the mechanisms described above:

- **Peer Discovery Protocol:** enables a peer to find advertisements on other peers, and can be used to find any of the peer, peer group, or advertisements. This protocol is the default discovery protocol for all peer groups, including the World Peer Group.
- **Peer Resolver Protocol:** enables a peer to send and receive generic queries to search for peers, peer groups, pipes, and other information.
- **Peer Information Protocol:** allows a peer to learn about the capabilities and status of other peers.
- **Peer Membership Protocol:** allows a peer to obtain group membership requirements, to apply for membership and receive a membership credential along with a full group advertisement, to update an existing membership or application credential, and to cancel a membership or an application credential.
- **Pipe Binding Protocol:** allows a peer to bind a pipe advertisement to a pipe endpoint, thus indicating where messages actually go over the pipe. In some sense, a pipe can be viewed as an abstract, named message queue that supports a number of abstract operations such as create, open, close, delete, send, and receive.

- **Peer Endpoint Protocol:** allows a peer to ask a peer router for available routes for sending a message to a destination peer.

JXTA supports the management of peer-to-peer communication of individual peers and peer groups, and enables communications among peers using pipes and eXtensible Markup Language (XML) documents. Figure 2.5. denotes the logical layers of JXTA, which support peer-to-peer communication:

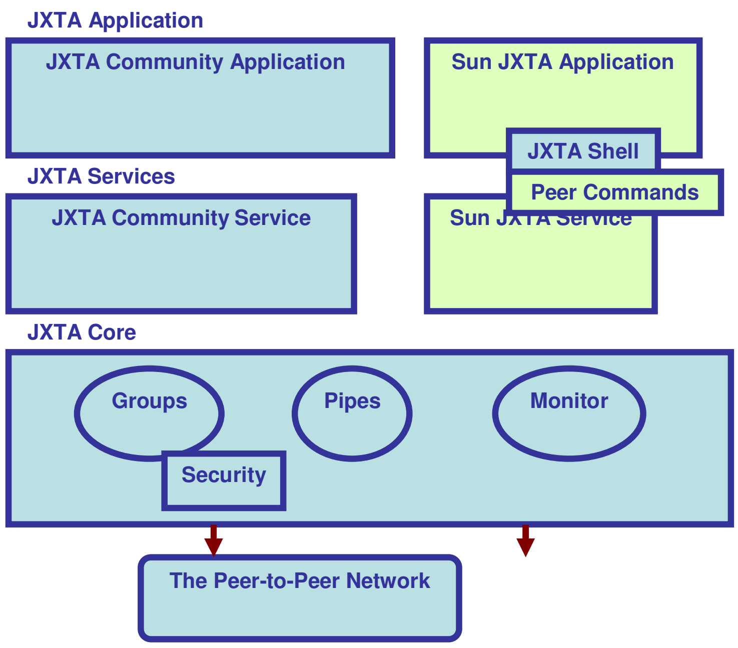


Figure 2.5. Logical Layers of JXTA

The Core layer consists of element peers, peer groups, entity names, and protocols, such as discovery, communication, and monitoring. This layer is the primary layer of the JXTA solution and also provides functionalities of the JXTA Peer-to-Peer (P2P) solution in the services or applications layers.

The Services layer performs network functions, such as communication between two peers or software components. This layer provides functionalities, such as sharing resources and documents on a peer and authenticating peers.

Services are built on top of the JXTA Core layer. These services provide the specific capabilities required by different P2P applications.
The Application layer is built above the Services layer. The system introduced witin this work has been developed in this layer.

2.3. Web Services

Web services are self-describing and modular applications that provide services over the Internet through programmable interfaces and using Internet protocols for the purpose of providing ways to find, subscribe, and invoke those services. Web services are autonomous platform-independent computational elements that can be described, published, discovered, orchestrated, and programmed using XML artefacts for the purpose of developing massively distributed interoperable applications.

Web services have been described as the new phase of the Internet. The emergence of Web services introduces a new paradigm for enabling the exchange of information across the Internet based on open Internet standards and technologies. Using industry standards, Web services encapsulate applications and publish them as services. These services deliver XML-based data for use on the Internet, which can be dynamically located, subscribed, and accessed using a wide range of computing platforms, handheld devices, appliances, and so on. Due to the flexibility of using open standards and protocols, it also facilitates Enterprise Application Integration (EAI), business-to-business (B2B) integration, and application-to-application (A2A) communication across the Internet and corporate intranet. In organizations with heterogeneous applications and distributed application architectures, the introduction of Web services standardizes the communication mechanism and enables interoperability of applications based on different programming languages residing on different platforms.

Based on XML standards, Web services can be developed as loosely coupled application components using any programming language, any protocol, or any platform. This facilitates delivering business applications as a service accessible to anyone, anytime, at any location, and using any platform.

As mentioned, Web services are implemented based on open standards and technologies specifically exploiting XML. The XML-based standards and technologies, such as Simple Object Access Protocol (SOAP); Universal Description, Discovery, and Integration (UDDI); Web Services Definition Language (WSDL); and Electronic Business XML (ebXML), are commonly used as building blocks for Web services, which will be described in the following sections.

Web services operations can be conceptualized as a simple operational model (Figure 2.6.). Operations are described as involving three distinct roles and relationships that define the Web services providers and users.

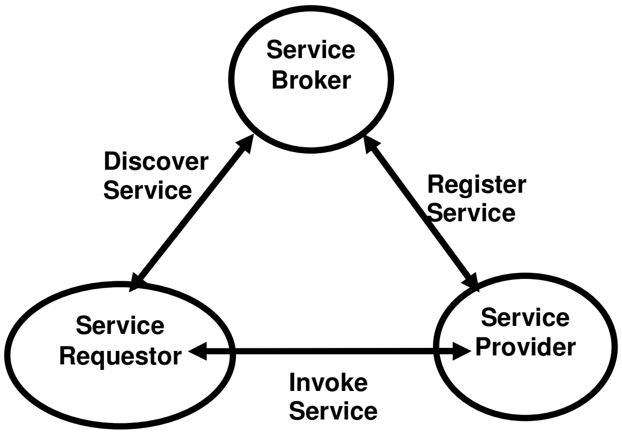


Figure 2.6. Web Services Operational Model

These roles and relationships are defined as follows:

- **Service Provider:** The service provider is responsible for developing and deploying the Web services. The provider also defines the services and publishes them with the service broker.
- **Service Broker:** The service broker (also commonly referred to as a service registry) is responsible for service registration and discovery of the Web services. The broker lists the various service types, descriptions, and locations of the services that help the service requesters find and subscribe to the required services.
- **Service Requestor:** The service requestor is responsible for the service invocation. The requestor locates the Web service using the service broker, invokes the required services, and executes it from the service provider.

The five core Web services standards and technologies for building and enabling Web services are XML, SOAP, WSDL, UDDI, and ebXML. An overview of each is presented in the following sections.

2.3.1. Extensible Markup Language (XML)

In February 1998, the Worldwide Web Consortium (W3C) officially endorsed the Extensible Markup Language (XML) as a standard data format. XML uses Unicode, and it is structured self-describing neutral data that can be stored as a simple text document for representing complex data and to make it readable. Today, XML is the de facto standard for structuring data, content, and data format for electronic documents. It has already been widely accepted as the universal language lingua franca for exchanging information between applications, systems, and devices across the Internet.

In the core of the Web services model, XML plays a vital role as the common wire format in all forms of communication. XML also is the basis for other Web services standards.

2.3.2. Simple Object Access Protocol (SOAP)

SOAP is a standard for a lightweight XML-based messaging protocol. It enables an exchange of information between two or more peers and enables them to communicate with each other in a decentralized, distributed application environment. Like XML, SOAP also is independent of the application object model, language, and running platforms or devices. SOAP is endorsed by W3C and key industry vendors like Sun Microsystems, IBM, HP, SAP, Oracle, and Microsoft. These vendors have already announced their support by participating in the W3C XML protocol-working group. The ebXML initiative from UN/CEFACT also has announced its support for SOAP.

In the core of the Web services model, SOAP is used as the messaging protocol for transport with binding on top of various Internet protocols such as HTTP, SMTP, FTP, and so on. SOAP uses XML as the message format, and it uses a set of encoding rules for representing data as messages. Although SOAP is used as a messaging protocol in Web services, it also can operate on a request/response model by exposing the functionality using SOAP/RPC based on remote procedural calls. SOAP also can be used with J2EE-based application frameworks.

2.3.3. Web Services Description Language (WSDL)

The Web Services Description Language (WSDL) standard is an XML format for describing the network services and its access information. It defines a binding mechanism used to attach a protocol, data format, an abstract message, or set of endpoints defining the location of services.

In the core of the Web services model, WSDL is used as the metadata language for defining Web services and describes how service providers and requesters communicate with one another. WSDL describes the Web services functionalities offered by the service provider, where the service is located, and how to access the service. Usually the service provider creates Web services by generating WSDL from its exposed business applications. A public/private registry is utilized for storing and publishing the WSDL-based information.

2.3.4. Universal Description, Discovery, and Integration (UDDI)

Universal Description, Discovery, and Integration, or UDDI, defines the standard interfaces and mechanisms for registries intended for publishing and storing descriptions of network services in terms of XML messages. It is similar to the yellow pages or a telephone directory where businesses list their products and services. Web services brokers use UDDI as a standard for registering the Web service providers. By communicating with the UDDI registries, the service requestors locate services and then invoke them.

In the core Web services model, UDDI provides the registry for Web services to function as a service broker enabling the service providers to populate the registry with service descriptions and service types and the service requestors to query the registry to find and locate the services. It enables Web applications to interact with a UDDI-based registry using SOAP messages. These registries can be either private services within an enterprise or a specific community, or they can be public registries to service the whole global business community of the Internet. The UDDI working group includes leading technology vendors like Sun Microsystems, IBM, HP, SAP, Oracle, and Microsoft.

2.3.5. ebXML

ebXML defines a global electronic marketplace where enterprises find one another and conduct business process collaborations and transactions. It also defines a set of specifications for enterprises to conduct electronic business over the Internet by establishing a common standard for business process specifications, business information modeling, business process collaborations, collaborative partnership profiles, and agreements and messaging. ebXML is an initiative sponsored by the United Nations Center for Trade Facilitation and Electronic Business (UN/CEFACT) and the Organization for the Advancement of Structured Information Standards (OASIS). Popular standards organizations like Open Travel Alliance (OTA), Open Application Group, Inc. (OAGI), Global Commerce Initiative (GCI), Health Level 7 (HL7, a healthcare standards organization), and RosettaNet (an XML standards committee) also have endorsed it.

In the Web services model, ebXML provides a comprehensive framework for the electronic marketplace and B2B process communication by defining standards for business processes, partner profile and agreements, registry and repository services, messaging services, and core components. It complements and extends with other Web services standards like SOAP, WSDL, and UDDI. In particular:

- ebXML Business Process Service Specifications (BPSS) enable business processes to be defined.
- ebXML CPP/CPA enables business partner profiles and agreements to be defined, and it provides business transaction choreography.
- ebXML Messaging Service Handler (MSH) deals with the transport, routing, and packaging of messages, and it also provides reliability and security, a value addition over SOAP.
- ebXML registry defines the registry services, interaction protocols, and message definitions, and ebXML repository acts as storage for shared information. The ebXML registries register with other registries as a federation, which can be discovered through UDDI. This enables UDDI to search for a business listing point to an ebXML Registry/Repository.
- ebXML Core components provide a catalogue of business process components that provide common functionality to the business

community. Examples of such components are Procurement, Payment, Inventory, and so on.

2.4. Web Services Security

Web services security proposes a standard set of SOAP extensions that can be used when building secure Web services to implement integrity and confidentiality. This set of extensions are referred as the “Web Services Security Language” or “WS-Security”.

WS-Security exploits the XML security technologies as the building bocks. XML-Encryption and XML-Digital Signature, which constitute the basis for XML security, are explained in the following sections.

2.4.1. XML Encryption

There are particular difficulties in dealing with hierarchical data structures, like XML, and with subsets of data with varying requirements as to confidentiality, access authority, or integrity. In addition, the application of standard security controls differentially to XML documents is not at all straightforward.

An XML document, like any other, can be encrypted in its entirety and sent securely to one or more recipients. This is a common function of Secure Sockets Layer (SSL) or Transport Layer Security (TLS), for example, but what is much more interesting is how to handle situations where different parts of the same document need different treatment. A valuable benefit of XML is that a complete document can be sent as one operation and then held locally, thus reducing network traffic. But this then raises the question of how to control authorized viewing of different groups of elements. In order to examplify with a possible scenario, a merchant may need to know a customer's name and

address but does not need to know the various details of any credit card being used any more than the bank needs to know the details of the goods bought. A researcher may need to be prevented from seeing personal details on medical records while an administrator may need exactly those details but should be prevented from viewing medical history; a doctor or nurse, in turn, may need medical details and some, but not all, personal material.
Cryptography now does far more than merely concealing information. Message digests confirm text integrity, digital signatures support sender authentication, and related mechanisms are used to ensure that a valid transaction cannot later be repudiated by another party.

One of the strengths of XML language is that searching is clear and unambiguous: The Document Type Definition (DTD) or schema provides information as to the relevant syntax. If a document subsection, including tags, is encrypted as a whole, then the ability to search for data relevant to those tags is lost. Further, if the tags are themselves encrypted, then, being known, they may be useful as material for mounting plain text attacks against the cryptography employed.

XML Encryption is a specification that has been introduced by W3C. XML Encryption specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption element which contains or references the cipher data.

The core element in the XML encryption syntax is the EncryptedData element which, with the EncryptedKey element, is used to transport encryption keys from the originator to a known recipient, and derives from the EncryptedType abstract type. Data to be encrypted can be arbitrary data, an XML document, an XML element, or XML element content; the result of encrypting data is an XML encryption element that contains or references the cipher data. When an element or element content is encrypted, the EncryptedData element replaces the element or content in the encrypted version of the XML document. When it is arbitrary data that is being encrypted, the `EncryptedData` element may become the root of a new XML document or it may become a child element. When an entire XML document is encrypted, then the EncryptedData element may become the root of a new document. Further, EncryptedData cannot be the parent or child of another EncryptedData element, but the actual data encrypted can be anything including existing EncryptedData or EncryptedKey elements.

In order to examplify the mechanism that XML Encryption introduces, if the following XML document is intended to be encrypted to conceal information on payment mechanisms:

```
       <?xml version='1.0'?>
       <PaymentInfo xmlns='http://example.org/paymentv2'>
         <Name>John Smith<Name/>
         <CreditCard Limit='5,000' Currency='USD'>
           <Number>4019 2445 0277 5567</Number>
           <Issuer>Bank of the Internet</Issuer>
           <Expiration>04/02</Expiration>
         </CreditCard>
       </PaymentInfo>
```

The resultant XML document becomes as follows:

```
       <?xml version='1.0'?>
       <PaymentInfo xmlns='http://example.org/paymentv2'>
         <Name>John Smith<Name/>
         <EncryptedData
Type='http://www.w3.org/2001/04/xmlenc#Element'
          xmlns='http://www.w3.org/2001/04/xmlenc#'>

<CipherData><CipherValue>A23B45C56</CipherValue></CipherData>
         </EncryptedData>
       </PaymentInfo>
```

In yet other cases, it might be necessary to conceal some sensitive content, which may result in the following XML document:

```
       <?xml version='1.0'?>
       <PaymentInfo xmlns='http://example.org/paymentv2'>
         <Name>John Smith<Name/>
         <CreditCard Limit='5,000' Currency='USD'>
```

```
    <Number>
      <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
       Type='http://www.w3.org/2001/04/xmlenc#Content'>
          <CipherData><CipherValue>A23B45C56</CipherValue>
          </CipherData>
      </EncryptedData>
    </Number>
    <Issuer>Bank of the Internet</Issuer>
    <Expiration>04/02</Expiration>
  </CreditCard>
</PaymentInfo>
```

2.4.2. XML Signature

Digital signatures provide end-to-end message integrity guarantees, and can also provide authentication information about the originator of a message. In order to be most effective, the signature must be part of the application data, so that it is generated at the time the message is created, and it can be verified at the time the message is ultimately consumed and processed.

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) also provide message integrity (as well as message privacy), but these mechanisms only do this while the message is in transit. Once the message has been accepted by the server (or, more generally, the peer receiver), the SSL protection must be "stripped off" so that the message can be processed. As a more subtle point, SSL and TLS only work between the communication endpoints.

An XML signature defines a series of XML elements that can be embedded in, or otherwise affiliated to, any XML document. It allows the receiver to verify that the message has not been modified from what the sender intended.

XML Signature specifies XML syntax and processing rules for creating and representing digital signatures. XML Signatures can be applied to any *digital content (data object)*, including XML. An XML Signature may be applied to the content of one or more resources.

Indeed, XML Signature is a method of associating a key with referenced data (octets); it does not normatively specify how keys are associated with persons or institutions, nor the meaning of the data being referenced and signed. Consequently, while this specification is an important component of secure XML applications, it itself is not sufficient to address all application security/trust concerns, particularly with respect to using signed XML (or other data formats) as a basis of human-to-human communication and agreement.
Prior to explaining the format and the mechanism that XML Signature introduces, the following paragraph details the process of achieving digital signatures:

A digital signature provides an integrity check on some content. If a single byte of the original content has been modified, then the signature fails to verify. The first step for creating a digital signature is to "hash" the message. A cryptographic hash takes an arbitrary stream of bytes and converts it to a single fixed-size value known as a *digest*. A digest is a one-way process: it is "computationally infeasible" to recreate a message from the hash, or to find two different messages which produce the same digest value.

If a message M is generated, and a digest is created, (written as **H(M)**, for "the hash of M"), and the receiver gets **M** and **H(M)**, the receiver can create his/her own digest **H'(M)**, and if the two digest values match, it is assured that the receiver gets what has originally been sent. In order to protect **M** against modification, it is only needed to protect **H(M)** from being modified.

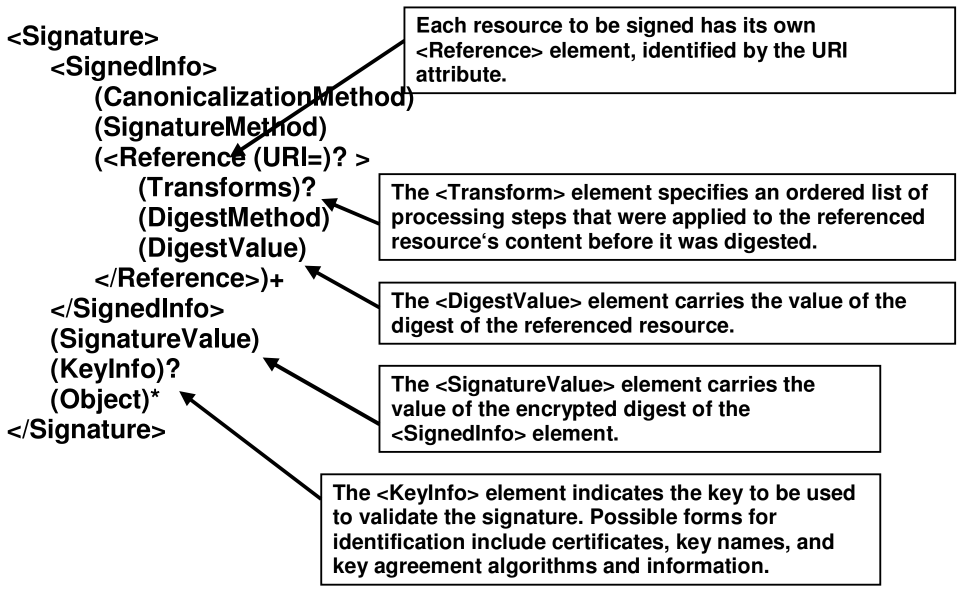
There are two common approaches to prevent **H(M)** from being modified. The first method is to mix a shared secret into the digest. In other words, to create **H(S+M)**. When the receiver gets the message, the receiver uses his/her own copy of **S** to create **H'(S+M)**. This new digest is called an HMAC, or Hashed Messsage Authentication Code.

Another method to protect the digest is to use public-key cryptography. In public-key cryptography, there are two keys, a private key, known only to the holder, and a public key, accessible to anyone who wants to communicate with the key holder. In public-key cryptography, anything encrypted with the private key can be decrypted with the public key, and vice versa. Using this method, a digest is generated, **H(M)**, and encrypted it with sender's private key, **{H(M)}private-key**, which is the signature. When the receiver gets the message, **M**, the receiver generates the digest, **H'(M)**, and decrypts the signature using the sender's public key, getting the **H(M)** that has been generated by the sender. If **H(M)** and **H'(M)** are the same, then it is assured that **M** is the same. Further, it is known that whoever has the private key is the sender of the message.

XML signatures are digital signatures designed for use in XML transactions. The standard defines a schema for capturing the result of a digital signature operation applied to arbitrary (but often XML) data.

A fundamental feature of XML Signature is the ability to sign only specific portions of the XML tree rather than the complete document. This will be relevant when a single XML document may have a long history in which the different components are authored at different times by different parties, each signing only those elements relevant to itself. This flexibility will also be critical in situations where it is important to ensure the integrity of certain portions of an XML document, while leaving open the possibility for other portions of the document to change. Consider, for example, a signed XML form delivered to a user for completion. If the signature is over the full XML form, any change by the user to the default form values invalidates the original signature.

The components of XML Signature is denoted in the following figure:



<Signature>
<SignedInfo>
(CanonicalizationMethod)
(SignatureMethod)
(<Reference (URI=)? >
(Transforms)?
(DigestMethod)
(DigestValue)
</Reference>)+
</SignedInfo>
(SignatureValue)
(KeyInfo)?
(Object)*
</Signature>
Each resource to be signed has its own <Reference> element, identified by the URI attribute.
The <Transform> element specifies an ordered list of processing steps that were applied to the referenced resource's content before it was digested.
The <DigestValue> element carries the value of the digest of the referenced resource.
The <SignatureValue> element carries the value of the encrypted digest of the <SignedInfo> element.
The <KeyInfo> element indicates the key to be used to validate the signature. Possible forms for identification include certificates, key names, and key agreement algorithms and information.

Figure 2.7. Components of XML Signature

The top-level Signature element covers information about what is being signed, the signature, and the keys used to create the signature.

In XML signatures, each referenced resource is specified through a Reference element and its digest (calculated on the identified resource and not the Reference element itself) is placed in a DigestValue child element. The DigestMethod element identifies the algorithm that is used to calculate the digest. Reference elements are collected with their associated digests within a SignedInfo element.

The CanonicalizationMethod element indicates the algorithm that is used to canonize the SignedInfo element. Different data streams with the same XML information set may have different textual representations, such as differing as to whitespace. In order to help prevent inaccurate verification results, XML information sets must first be canonized before extracting their bit representation for signature processing. The SignatureMethod element identifies the algorithm used to produce the signature value.

The digest of the SignedInfo element is calculated, that digest is signed, and the signature value is put in a SignatureValue element.

If the keying information is to be included, it is placed in a KeyInfo element. The SignedInfo, SignatureValue, and KeyInfo elements are placed into a Signature element. The Signature element comprises the XML signature.

The following is a sample XML signature document:

```
<?xml version="1.0" encoding="UTF-8"?>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo Id="foobar">
<CanonicalizationMethod
  Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
<SignatureMethod
  Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"
/>
<Reference
URI="http://www.abccompany.com/news/2000/03_27_00.htm">
<DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
</Reference>
<Reference
  URI="http://www.w3.org/TR/2000/WD-xmldsig-core-
20000228/signature-example.xml">
<DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<DigestValue>UrXLDLBIta6skoV5/A8Q38GEw44=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>MC0E~LE=</SignatureValue>
<KeyInfo>
<X509Data>
<X509SubjectName>CN=Ed Simon,O=XMLSec
Inc.,ST=OTTAWA,C=CA</X509SubjectName>
<X509Certificate>
MIID5jCCA0+gA...lVN
</X509Certificate>
</X509Data>
</KeyInfo>

</Signature>
```

2.4.3. WS-Security

Web Services Security (WS-Security) describes enhancements to SOAP messaging to provide *quality of protection* through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

WS-Security is flexible and is designed to be used as the basis for the construction of a wide variety of security models, such as Public Key Infrastructure (PKI). Specifically WS-Security provides support for multiple security tokens, multiple trust domains, multiple signature formats, and multiple encryption technologies.

WS-Security provides three main mechanisms:

- security token propagation,
- message integrity, and
- message confidentiality.

These mechanisms by themselves do not provide a complete security solution. Instead, WS-Security is a building block that can be used in conjunction with other Web service extensions and higher-level application-specific protocols to accommodate a wide variety of security models and encryption technologies. WS-Security specifies the methods to embed security within the SOAP message itself, and addresses: authentication, signatures, and encryption.

WS-Security addresses concerns related with security by leveraging existing standards and specifications, such as XML Encryption, XML Signature, and XML Canonicalization. This avoids the necessity to define a complete security solution within WS-Security. What WS-Security adds to existing specifications is a framework to embed these mechanisms into a SOAP message.

WS-Security defines a SOAP Header element to carry security-related data. If XML Signature is used, this header can contain the information defined by XML Signature that conveys how the message is signed, the key that is used, and the resulting signature value. Likewise, if an element within the message is encrypted, the encryption information such as that conveyed by XML Encryption can be contained within the WS-Security Header element. WS-Security does not specify the format of the signature or encryption. Instead, it specifies how one would embed the security information laid out by other specifications within a SOAP message. WS-Security is primarily a specification for an XML-based security metadata container.

Figure 2.8. depicts a fairly common message flow:

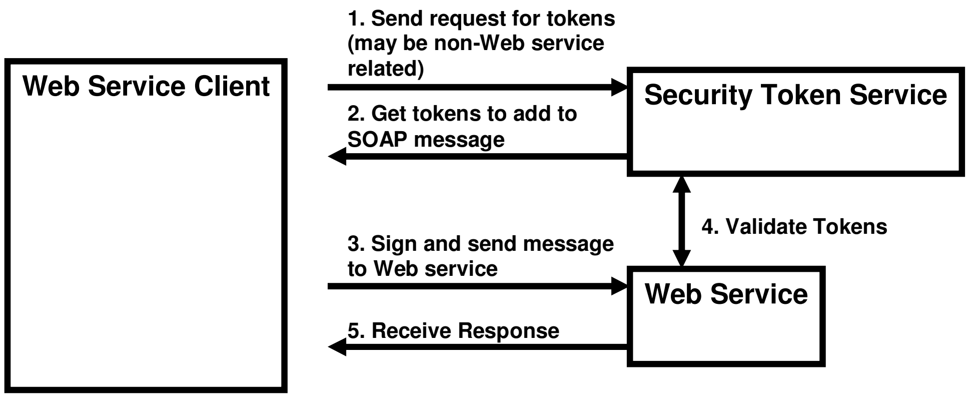

Web Service Client
1. Send request for tokens (may be non-Web service related)
Security Token Service
2. Get tokens to add to SOAP message
4. Validate Tokens
3. Sign and send message to Web service
Web Service
5. Receive Response

Figure 2.8. Typical Message Flow

2.5. XML Key Management Specification (XKMS)

The XML Key Management Specification (XKMS) defines processes and formats that enable XML-aware applications to incorporate the security of public-key infrastructure (PKI).

XKMS defines two classes of PKI functions:

- Public key lifecycle management (registration, revocation and, renewal),
- Validation and location of cryptographic keys.

XKMS provides an alternative approach to developers implementing PKI-enabled applications. Public Key Infrastructure (PKI) is essential for enabling trust in digital communications.

PKI enables to digitally sign documents, and then to verify those signatures. PKI also enables to encrypt and decrypt documents. However, complex and expensive infrastructure is required to perform these seemingly simple tasks. Many hours must be spent by developers integrating applications, such as e-mail clients or ERP systems, with a PKI. And once one ERP or e-mail client is PKI-enabled, it does not necessarily function with another PKI-enabled ERP or another e-mail client.

XKMS solves the problem of complexity by enabling client applications to delegate processing to server-based services. With XKMS, PKI complexity is hidden in the infrastructure, making it easy to integrate cryptographic trust services into more applications. At the same time, XKMS simplifies and standardizes the interface between a client application and a PKI. Standards-based interfaces make interoperable PKI feasible. In other words, XKMS makes it possible simply to "plug in" to a trust service that enables applications to easily perform cryptographic functions.

With XKMS, trust functions reside in servers accessible via easily programmed XML transactions. XKMS also enables increased interoperability between XKMS systems by combining XML with PKI.

Incorporating digital signatures or encryption functionality into applications help in gaining the following benefits when XKMS is used:

- Developers can integrate authentication, digital signature, and encryption services, such as revocation status checking into applications doing away with the constraints and complications associated with proprietary PKI software toolkits.
- XKMS provides a *pure XML*, developer-friendly syntax. By avoiding the introduction of PKI toolkits on the client side, the trust infrastructure permits developers to work in a simple XML environment.
- XKMS enables rapid implementation of trust with standard XML toolkits. The only client logic beyond traditional XML parsing runtimes is cryptographic support for XML digital signatures and XML encryption.
- XKMS does not require proprietary plug-ins to support enterprise PKI.
- XKMS reduces delays in PKI deployment, as it moves the complexity of PKI and trust processing from client-side to server-side components.

2.6. WS-Trust

The Web Services Security (WS-Security) roadmap describes mechanism for addressing security within a Web service environment. It defines a comprehensive Web service security model that supports, integrates, and unifies several popular security models, mechanisms, and technologies (including both symmetric and public key technologies) in a way that enables a variety of systems to securely interoperate in a platform- and language-neutral manner. It describes scenarios that show how the specifications like WS-Trust and WS-SecurityPolicy might be used together.

Figure 2.9. denotes the WS-Security specification as the building block for the specifications WS-Trust and WS-SecurityPolicy.

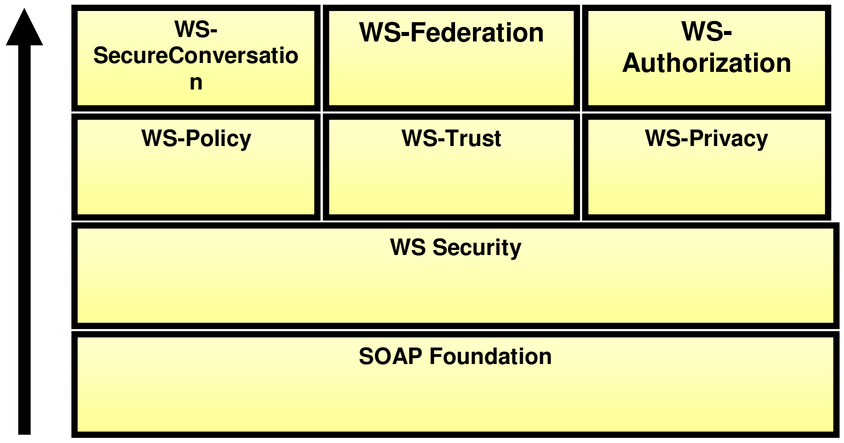


Figure 2.9. Web Services Specifications

WS-Trust describes the model for establishing both direct and brokered trust relationships including third parties and intermediaries. This specification defines extensions that build on WS-Security to provide a framework for requesting and issuing security tokens, and to broker trust relationships.

WS-Security defines the basic mechanisms for providing secure messaging. WS-Trust uses these base mechanisms and defines additional primitives and extensions for security token exchange to enable the issuance and dissemination of credentials within different trust domains.

WS-Trust defines extensions to WS-Security that provide:

- Methods for issuing, renewing, and validating security tokens.
- Ways to establish, assess the presence of, and broker trust relationships.

Using these extensions, applications can engage in secure communication designed to work with the general Web services framework, including WSDL service descriptions, UDDI businessServices and bindingTemplates, and SOAP messages. To achieve this, WS-Trust introduces a number of elements that are used to request security tokens and broker trust relationships.

The goal of WS-Trust is to enable applications to construct trusted SOAP message exchanges. This trust is represented through the exchange and brokering of security tokens. WS-Trust provides a protocol agnostic way to issue, renew, and validate these security tokens.

WS-Trust intends to provide a flexible set of mechanisms that can be used to support a range of security protocols; that is, WS-Trust intentionally does not describe explicit fixed security protocols.

The Web service security model defined in WS-Trust is based on a process in which a Web service can require that an incoming message prove a set of claims (e.g., name, key, permission, capability, etc.). If a message arrives without having the required proof of claims, the service ignores or rejects the message. A service can indicate its required claims and related information in its policy as described by WS-SecurityPolicy specification.

The model introduced by WS-Trust is illustrated in the Figure 2.10., which denotes that any requestor may also be a service, and that the Security Token Service is a Web service; that is, it may express policy and require security tokens.

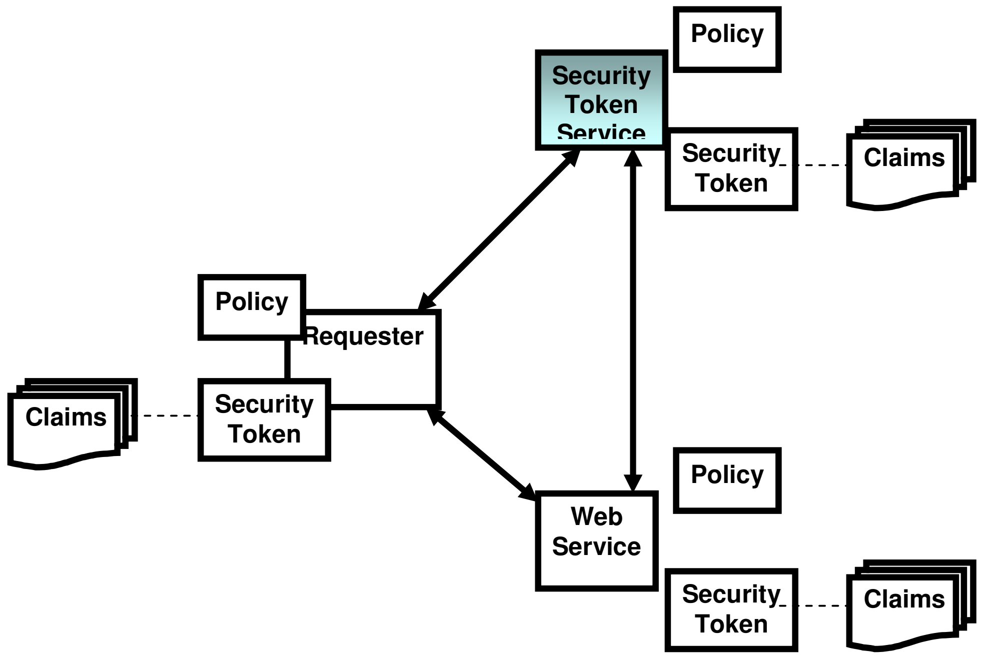


Figure 2.10. Security Token Service Model

WS-Trust defines what requests to a security token service look like and how responses are sent back. The interaction relies on two elements: RequestSecurityToken and RequestSecurityTokenResponse. The basic idea is to send a SOAP message containing a RequestSecurityToken as its body, then get back a SOAP message containing a RequestSecurityTokenResponse, complete with the new security token.
A sample body of the response returned by the security token service might look like:

<s:Body> <wsse:RequestSecurityTokenResponse> <wsse:RequestedSecurityToken> <wsse:BinarySecurityToken ValueType="wsse:X509v3" EncodingType="wsse:Base64Binary"> KkFPle ... </wsse:BinarySecurityToken> </wsse:RequestedSecurityToken> </wsse:RequestSecurityTokenResponse> </s:Body>

2.7. WS-SecurityPolicy

The Web Services Policy Framework (WS-Policy) provides a general purpose model and corresponding syntax to describe and communicate the policies of a Web Service.

WS-Policy defines a base set of constructs that can be used and extended by other Web Services specifications to describe a broad range of service requirements, preferences, and capabilities.

In order to successfully integrate with a nontrivial Web service, the service's XML contract along with any additional requirements, capabilities, and preferences, also referred to as policies, must fully be understood. For example, just knowing that a service supports WS-Security is not enough information to enable successful integration. The client needs to know if the service actually requires WS-Security. If so, it also needs to know what security tokens it is capable of processing (such as UsernameToken, or certificates), and which one it prefers. The client must also determine if the service requires signed messages. And if so, it must determine what token type must be used for the digital signatures. And finally, the client must determine when to encrypt the messages, which algorithm to use, and how to exchange a shared key with the service. Trying to integrate with a service without understanding these details is a stab in the dark.

A standard policy framework would make it possible for developers to express the policies of services in a machine-readable way. Web services infrastructure could be enhanced to understand certain policies and enforce them at runtime. For example, a developer could write a policy stating that a given service requires Kerberos tokens, digital signatures, and encryption. Other developers could use the policy information to reason about whether it can use the service. Plus, the infrastructure could enforce these requirements without requiring the developer to write a single line of code. So not only would a policy framework provide an additional description layer, it would also offer developers a more declarative programming model.

Microsoft, IBM, BEA, and SAP has released a specification called the Web Services Policy Framework (WS-Policy) to fill the need for a generic policy framework. WS-Policy defines a generic model and syntax for describing and communicating the policies of a Web service.

WS-Policy defines a general framework that can be used and extended by other Web services specifications to describe a broad range of Web services policies. WS-Policy defines a policy to be a collection of one or more policy assertions (Figure 2.11.).

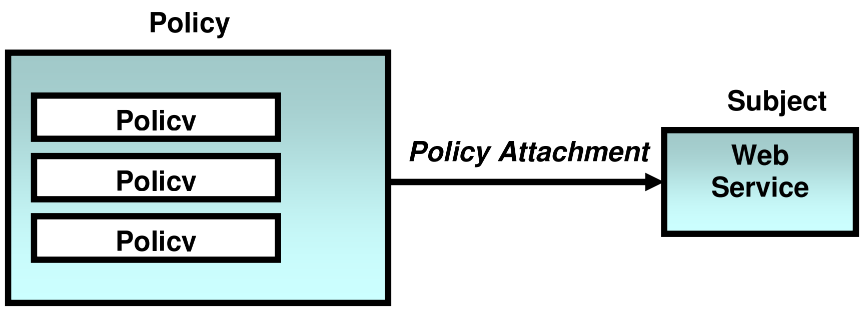


Figure 2.11. Policy Overview

A policy assertion represents an individual preference, requirement, capability, or other general characteristic. There are two additional specifications that define standard sets of policy assertions that can be used within a policy expression. The Web Services Policy Assertions Language (WS-PolicyAssertions) specification defines a set of general message assertions and the Web Services Security Policy Language (WS-SecurityPolicy) specification defines a set of common security-related assertions.

The WS-Policy specification defines the general model and syntax for policy expressions and policy assertions, but stops short of specifying how policies are located or attached to a Web service.

A policy assertion represents an individual preference, requirement, capability, or other characteristic and is the basic building block of a policy expression. A policy assertion is represented by an XML element with a well-known name and meaning, typically defined by another specification like WS-SecurityPolicy.

A policy expression simply contains a set of policy assertion elements as follows:

```
<wsp:Policy xmlns:wsp="..." xmlns:wsu="..." wsu:Id="..." Name="..." TargetNamespace="..."
> <Assertion wsp:Usage="..." wsp:Preference="..." /> <Assertion wsp:Usage="..."
wsp:Preference="..." /> <Assertion wsp:Usage="..." wsp:Preference="..." /> ... </wsp:Policy>
```

WS-SecurityPolicy specification defines a set of security-related policy assertions (Table 2.1.). These assertions allow to specify the types of security tokens, signature formats, and encryption algorithms supported, required, or rejected by a given subject.

The SecurityToken element is used to describe what security tokens are required and accepted by a Web service. It can also be used to express a Web Service's policy on security tokens that are included when the service sends out a message (e.g., as a reply message).

```
<SecurityToken wsp:Preference="..." wsp:Usage="..." > <TokenType>...</TokenType>
<TokenIssuer>...</TokenIssuer> <Claims>...Token type-specific claims...</Claims> ...
(TokenType-specific details) </SecurityToken>
```

Table 2.1. Security Policy Assertions defined by WS-SecurityPolicy

Policy Assertion	Policy Assertion Description
wsse:SecurityToken	specifies a type of security token defined in WS-Security
wsse:Integrity	specifies a signature format defined in WS-Security
wsse:Confidentiality	specifies an encryption format defined in WS-Security
wsse:Visibility	specifies portions of a message that MUST be able to be processed by an intermediary or endpoint
wsse:SecurityHeader	specifies how to use the Security header defined in WS-Security
wsse:MessageAge	specifies the acceptable time period before messages are declared “stale” and discarded

The Integrity element is used to indicate a required signature format. The schema outline for Integrity, an assertion about an integrity requirement, is as follows:

<Integrity wsp:Preference="..." wsp:Usage="..."> <Algorithm Type="..." URI="..." wsp:Preference="..."/> <TokenInfo> <SecurityToken>...</SecurityToken> </TokenInfo> <Claims>...</Claims> <MessageParts Dialect="..." Signer="..."> ... </MessageParts> <Integrity>

The Confidentiality element is used to indicate a required encryption format. The schema outline for this element is as follows:

<Confidentiality wsp:Preference="..." wsp:Usage="..."> <Algorithm Type="..." URI="..." wsp:Preference="..."/> <KeyInfo> <SecurityToken .../> <SecurityTokenReference .../> ... </KeyInfo> <MessageParts Dialect="..."> ... </MessageParts> </Confidentiality>

Some intermediaries may require that parts of the message be visible to them. That is, they either need to be passed in the clear (unencrypted), or there must be an encryption binding for the intermediary. The Visibility element is used to indicate portions of a message that MUST be able to be processed by an intermediary or endpoint. The schema outline for this element is as follows:

```
<Visibility wsp:Usage="..."> <MessageParts Dialect="..."> ... </MessageParts> </Visibility>
```

CHAPTER III

SECURE WEB SERVICES IN PEER-TO-PEER ENVIRONMENT

In this chapter, the approach and the mechanisms that are pursued in this work in order to provide and invoke secure Web services in peer-to-peer environment are descibed.

For this purpose, the specifications like WS-Security, WS-Trust, WS-SecurityPolicy, and XML Key Management Specification are exploited in addition to JXTA framework that has been benefitted from for peer-to-peer network implementation.

A peer-to-peer network, over JXTA framework, that consists of at least three peers is introduced in this work. The functionalities and responsibilities of the peers within this network are defined in Table 3.1.

Considering the peer groups that are described in Table 3.1., the demonstrational peer-to-peer network covers three peers, which is adequate to justify the proposed architecture.

Table 3.1. Peer-to-peer Network Elements

Peer-to-peer Network Groups	**Functionality**
PeerGroup-1	- provides Web service to be invoked by peer-2 - processes secure SOAP message by decrypting and/or verifying digital signature - validates token information available in the SOAP message - invokes XML Key Management Web service, that is provided by peer-3, to locate the public key information - invokes security token Web service, that is provided by peer-3, to locate and validate token information - invokes XML Key Management Web service, that is provided by peer-3, to locate the secret key
PeerGroup-2	- invokes Web service provided by peer-1 - provides the capability of constructing the SOAP message from the scratch by parsing the WSDL file for the Web service of peer-1 - encrypts the depicted element of the constructed SOAP message with the secret key that is located by invoking the XML Key Management Web service provided by peer-3 - digitally signs the depicted element of the constructed SOAP message with the private key that is kept by peer-2 - adds the token information, that is located by invoking the security token Web service provided by peer-3, to the constructed SOAP message - performs secure invocation of Web service provided by peer-1
PeerGroup-3	- provides XML Key Management Web service for peer-1 and peer-2 - provides security token Web service for peer-1 and peer-2

The interaction among the peers belonging to the peer groups described in Table 3.1 is depicted in Figure 3.1, which also denotes the overall architecture of the introduced peer-to-peer network.

The three peer groups that constitute the basis for the peer-to-peer network architecture proposed in this work are described in the following sections.

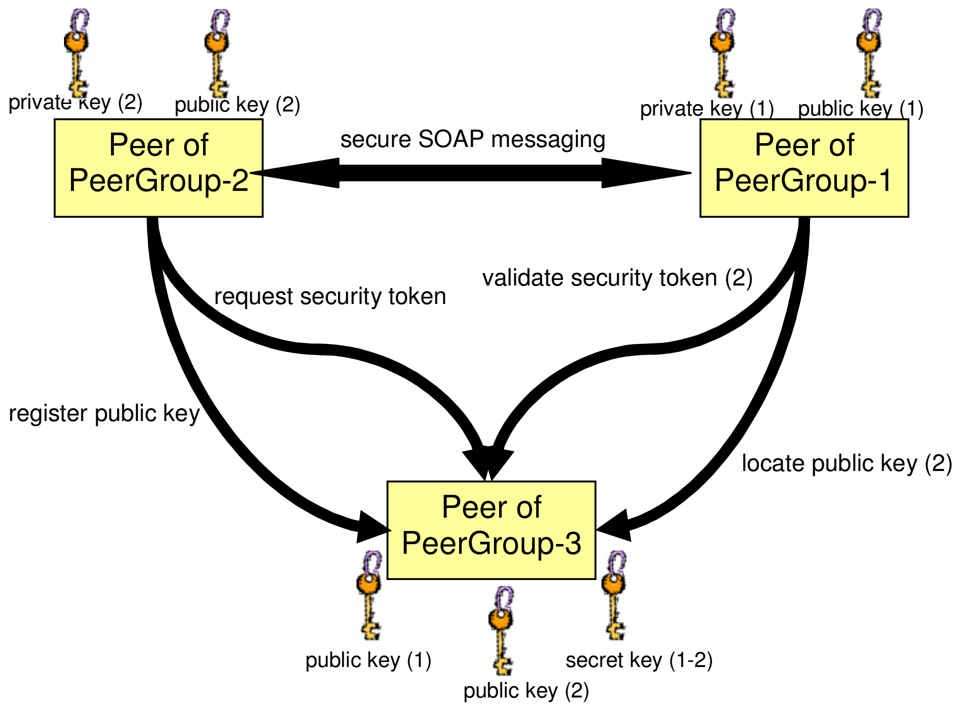


Figure 3.1. Interaction Diagram within the P2P Network

3.1. Description of Peers in PeerGroup-1

The peers of PeerGroup-1 conform to the specifications described by SOAP, WSDL, XKMS, WS-Trust, WS-Security, and WS-SecurityPolicy in order to provide secure Web service implementation. These peers process the security token information, encryption and the digital signatures prior to evaluating the response of their services.

The Web Services Description Language (WSDL) files for the Web services provided by the peers of this group are assumed to be available for the peers of PeerGroup-2 within the peer-to-peer network.

In addition, the security policy assertions (WS-SecurityPolicy) files for the Web services provided by the peers of this group are assumed to be available for the peers of PeerGroup-2 within the peer-to-peer network. The following assertions examplify a possible content of such a file that conforms to the specification WS-SecurityPolicy:

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
        <wsse:SecurityToken wsp:Usage="wsp:Rejected">
                <wsse:TokenType>wsse:Kerberosv5ST</wsse:TokenType>
        </wsse:SecurityToken>
        <wsse:SecurityToken wsp:Usage="wsp:Required">
                <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
        </wsse:SecurityToken>
        <wsse:SecurityToken wsp:Usage="wsp:Optional">
                <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
        </wsse:SecurityToken>
        <wsse:Integrity wsp:Usage="wsp:Required">
                <wsse:Algorithm Type="wsse:AlgSignature" />
                <MessageParts> xxx </MessageParts>
        </wsse:Integrity>
        <wsse:Confidentiality wsp:Usage="wsp:Required">
                <wsse:Algorithm Type="wsse:AlgEncryption" />
                <MessageParts> yyy </MessageParts>
        </wsse:Confidentiality>
</wsp:Policy>
```

The XML Key management specification is exploited by the peers of PeerGroup-1 in order to locate the public key information, which is required for the verification of digital signature available in the SOAP message, by invoking the XML Key Management Web service provided by the peers of

PeerGroup-3. It is assumed that the WSDL file for this XML Key Management service is available for the peers of this group. The following SOAP message examplifies a possible content of a SOAP message that requests the location of a public key information and conforms to the XML Key Management Specification:

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

<SOAP-ENV:Envelope                                          xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <k:Locate xmlns:k="http://www.xkms.org/schema/xkms-2001-01-20">
      <k:TransactionID>c41696b0-1f14-11d6-b840-
3beb2501bc66</k:TransactionID>
      <k:Query>
        <d:KeyInfo xmlns:d="http://www.w3.org/2000/09/xmldsig#">

<d:KeyName>http://xkms.verisign.com/key?company=VeriSign&amp;department=X
KMS                                              Test&amp;CN=N100055
XKMSTEST&amp;issuer_serial=85dfd14e8c4ecb4adeec353ca4c7b196</d:KeyName
>
        </d:KeyInfo>
      </k:Query>
      <k:Respond>
        <k:string>KeyName</k:string>
        <k:string>KeyValue</k:string>
        <k:string>X509Chain</k:string>
      </k:Respond>
      <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"/>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
sha1"/>
          <ds:Reference URI="#xpointer(/k:Locate)">
            <ds:Transforms>
```

```
                 <ds:Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                 <ds:Transform          Algorithm="http://www.w3.org/TR/2001/REC-xml-
c14n-20010315"/>
               </ds:Transforms>
               <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

<ds:DigestValue>YVwLOtUSstnIDPvFQXOZbg+ifC4=</ds:DigestValue>
             </ds:Reference>
           </ds:SignedInfo>

<ds:SignatureValue>GuzjcpjLjSH1doWh6cdz16HZeY3F7z0OvU5J8TR7kG13Zdo2b
L9f/BI3tJmCQz32T+R1MBxY
hJ4zX3gKlfKwRQ==</ds:SignatureValue>
           <ds:KeyInfo>
             <ds:KeyValue>
               <ds:RSAKeyValue>

<ds:Modulus>uabFEH3e0QBcQPbo3ixRGZ+GpJqaUjs+P4+JzhmIXmhI0SmpM1iw
910Egu1IO6y0lze6g6KU
ob7LSrO4/bIBHQ==</ds:Modulus>
                 <ds:Exponent>Aw==</ds:Exponent>
               </ds:RSAKeyValue>
             </ds:KeyValue>
           </ds:KeyInfo>
         </ds:Signature>
       </k:Locate>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The specification WS-Trust is exploited by the peers of PeerGroup-1 in order to locate the token information, which is required for the validation of security token information available in the SOAP message, by invoking the security token Web service provided by the peers of PeerGroup-3. The following SOAP message examplifies a possible content of a SOAP message that requests the location of username token information and conforms to the specification WS-Trust:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
 xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
 <s:Body>
  <wsse:RequestSecurityToken>
   <wsse:TokenType>wsse:UserName</wsse:TokenType>
   <wsse:RequestType>wsse:ReqIssue</wsse:RequestType>
   <wsse:Base>
    <wsse:Reference URI="#peer2"/>
   </wsse:Base>
  </wsse:RequestSecurityToken>
 </s:Body>
</s:Envelope>
```

3.2. Description of Peers in PeerGroup-2

The peers of PeerGroup-2 conform to the specifications described by SOAP, WSDL, XKMS, WS-Trust, WS-Security, and WS-SecurityPolicy in order to perform the secure invocation of the Web service provided by the peers of PeerGroup-1.

The Web Services Description Language (WSDL) files for the Web services provided by the peers of PeerGroup-1 are assumed to be available for the peers of this group. Hence, these peers enable the capability of providing a user interface to construct the related SOAP messages from the scratch by parsing the related WSDL files, which is to be demonstrated in more detail in the next chapter.

Having constructed the SOAP messages with the capability mentioned above, these peers achieve the insertion of digital signatures on the depicted SOAP message elements conforming to the specification WS-Security. Similarly, the peers of PeerGroup-2 achieve the encryption of the depicted SOAP message elements conforming to the specification WS-Security.

The XML Key management specification is exploited by the peers of PeerGroup-2 in order to locate the secret key information, which is required for the encryption of the depicted SOAP message elements, by invoking the security token Web service provided by the peers of PeerGroup-3. It is assumed that the WSDL file for this security token service is available for the peers of this group. The following SOAP message examplifies a possible content of a SOAP message that requests the location of a secret key information for the connection between the peers of PeerGroup1 and the peers of PeerGroup2:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
 xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
 <s:Body>
  <RequestSecretKey>
   <Party1>peer1</Party1>
   <Party2>peer2</Party2>
  </RequestSecretKey>
 </s:Body>
</s:Envelope>
```

The peers of PeerGroup-2 achieve the insertion of digitally signature, that is applied to the constructed SOAP message, by using the private keys of these peers. The sample content of how the private keys are stored by these peers can be denoted as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds:KeyInfo
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:KeyValue>
        <xkms:RSAKeyPair
xmlns:xkms="http://www.xkms.org/schema/xkms-2001-01-20">


<xkms:Modulus>AM2EJMuJrLtDGDDO3d56SoWqEXcvWJFXGAKrmexdnbd
xgskTeHYat7993HzrvCUQsNg1U3LV

0jEceHawgNCUB+sDV3pFYrrzK0vzkvdtHrILC+6ucztLeYx8NhVU9hXIt
5bqiQxOD4P/YRW0
LNSU55mX0bJIYybSqW/DqeWoXJrX</xkms:Modulus>

<xkms:PublicExponent>AQAB</xkms:PublicExponent>


<xkms:PrivateExponent>A7dM9wUxQ12ONWu4JUquyEXv9Oi9QDuD6qB
xsw1qSaPayon4lqIbCnJbJ0MX7s+jcvVb4eqt

10d9gh/k5GCY2AWtOtiTnFrpuPgIo91JZKNc8YuiVy/3ZZTvhE6T5fVVT
9rgcsGuIpc+YUPr
iKHUvuylKEyTjc2A5DlqvwwErcE=</xkms:PrivateExponent>


<xkms:P>APxlVVXNqDp42Y1MWlp1DDpw4fLnjPPT/9X/+T1S7HWnZQyeb
eKoGI9Tke73687msbDr2aST c09tN4L0HY9L8qE=</xkms:P>


<xkms:Q>ANBzb4Ji99H0y3VEQlrDJ+t79eycI13JE5k6YMVIvlZ2Hz9IP
ZvRplVFW4Xzw4lutC93rlrS 3B1VVqWarVxPknc=</xkms:Q>


<xkms:DP>AObVPqCEwJEJqd5isFf+qLpiNyPSxcTSZS8xNp/xUTWhbPeH
hfQ/zIZ45gTqVY4ayvSGH702 5cUEnDp2TPqbeUE=</xkms:DP>


<xkms:DQ>afUTjIKEGAxH037z/7DNvOyQ8EnZzFVmie/busykO8zlS0SM
VIU3+IY95JQpI9XK74n1baNb UenMFnfOgWrNNw==</xkms:DQ>


<xkms:QINV>AKnDNGMZJtCrdmzEoCrURRQDBlkOEBIj3OvQFuZijopGhx
ILXQHTpmucEpdaIFNH4sXfiYi8 oOAGJKTqNJzeeUI=</xkms:QINV>
        </xkms:RSAKeyPair>
    </ds:KeyValue>
</ds:KeyInfo>
```

In addition, the security policy assertions (WS-SecurityPolicy) files for the Web services provided by the peers of PeerGroup-1 are assumed to be available for the peers of this group, because the WS-SecurityPolicy specification does not currently address the specifying how these security policies are located or attached to a Web service. The peers of this group apply the encryption and/or digital signature operations conforming to the security assertions that are defined in WS-SecurityPolicy files assumed to be available, like the WSDL files.

3.3. Description of Peers in PeerGroup-3

The peers of PeerGroup-3 provide two key services for the other two peer groups. The two services are:

- XML Key Management Service, and
- Security Token Service.

These two Web services are detailed in the following subsections.

3.3.1. XML Key Management Service

XML Key Management service conforms to the XML Key Management Specification (XKMS), which defines processes and formats that enable XML-aware applications to incorporate the security of public-key infrastructure (PKI).

XML Key Management service performs the following functions as a service for the peers of other two peer groups:

- Public key lifecycle management (registration, revocation and, renewal),

  Following is a sample content of a SOAP message that requests the revocation of a specified public key information:

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

```
<SOAP-ENV:Envelope                                        xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <k:Register xmlns:k="http://www.xkms.org/schema/xkms-2001-01-20">
      <k:Prototype Id="refId_1">
        <k:TransactionID>cde285f0-1f14-11d6-b840-
3beb2501bc66</k:TransactionID>
        <k:Status>Invalid</k:Status>
        <d:KeyInfo xmlns:d="http://www.w3.org/2000/09/xmldsig#">
          <d:KeyValue xmlns:d="http://www.w3.org/2000/09/xmldsig#">
            <d:RSAKeyValue>
<d:Modulus>rlc2VJ+acRniqpF8dh5N2qyUYOkne5C257v5Rme13KGl4B7S
mAa340uBBicdx2RkfyHpW0q3Nm/iY7h8UY1+Cw==</d:Modulus>
              <d:Exponent
xmlns:d="http://www.w3.org/2000/09/xmldsig#">AQAB</d:Exponent>
            </d:RSAKeyValue>
          </d:KeyValue>
        </d:KeyInfo>
        <k:PassPhrase>SxbPA0qih0NsrYU8ji+gWOhwy7c=</k:PassPhrase>
      </k:Prototype>
      <k:AuthInfo>
        <k:AuthUserInfo/>
      </k:AuthInfo>
      <k:Respond>
        <k:string>KeyName</k:string>
        <k:string>KeyValue</k:string>
      </k:Respond>
    </k:Register>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

- Validation and location of cryptographic keys.

  Following is a sample content of a SOAP message that requests the validation of a specified public key information:

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

```
<SOAP-ENV:Envelope                                              xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <k:Validate xmlns:k="http://www.xkms.org/schema/xkms-2001-01-20">
      <k:Query>
        <k:TransactionID>cb618060-1f14-11d6-b840-
3beb2501bc66</k:TransactionID>
        <k:Status>Indeterminate</k:Status>
        <d:KeyInfo xmlns:d="http://www.w3.org/2000/09/xmldsig#">
          <d:KeyValue xmlns:d="http://www.w3.org/2000/09/xmldsig#">
            <d:RSAKeyValue>

<d:Modulus>rlc2VJ+acRniqpF8dh5N2qyUYOkne5C257v5Rme13KGl4B7S
mAa340uBBicdx2RkfyHpW0q3Nm/iY7h8UY1+Cw==</d:Modulus>
              <d:Exponent
xmlns:d="http://www.w3.org/2000/09/xmldsig#">AQAB</d:Exponent>
            </d:RSAKeyValue>
          </d:KeyValue>
        </d:KeyInfo>
      </k:Query>
      <k:Respond>
        <k:string>KeyName</k:string>
        <k:string>KeyValue</k:string>
        <k:string>ValidityInterval</k:string>
        <k:string>KeyUsage</k:string>
        <k:string>Status</k:string>
        <k:string>X509Chain</k:string>
      </k:Respond>
      <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"/>
          <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="#xpointer(/k:Validate)">
            <ds:Transforms>
```

```
              <ds:Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
              <ds:Transform   Algorithm="http://www.w3.org/TR/2001/REC-
xml-c14n-20010315"/>
            </ds:Transforms>
            <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

<ds:DigestValue>nM21iuKLqO0yUcdT5vYybfyDofA=</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>

<ds:SignatureValue>ttGw0FexCEJjzNbws29NbCU2oqdUpPOsFGasXZ4M
X3ZQKLEgV35DG9hmJdenxSGLFOvVulEv
sTLPKIU+0tEhSQ==</ds:SignatureValue>
        <ds:KeyInfo>
          <ds:KeyValue>
            <ds:RSAKeyValue>

<ds:Modulus>uabFEH3e0QBcQPbo3ixRGZ+GpJqaUjs+P4+JzhmIXmhI0S
mpM1iw910Egu1IO6y0lze6g6KU
ob7LSrO4/bIBHQ==</ds:Modulus>
              <ds:Exponent>Aw==</ds:Exponent>
            </ds:RSAKeyValue>
          </ds:KeyValue>
        </ds:KeyInfo>
      </ds:Signature>
    </k:Validate>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

With the functionalities described above, XML Key Management service provides an interface for implementing Public Key Infrastructure (PKI), which is essential for enabling trust in digital communications. Thus, the peers of this group behave as a trusted third party for the peers of PeerGroup-1 and PeerGroup-2.

3.3.2. Security Token Management Service

Security token service conforms to the specification WS-Trust, which defines extensions built on WS-Security to provide a framework for requesting and issuing security tokens, and to broker trust relationships.

Security token service performs the following functions as a service for the peers of other two peer groups:

- location of security tokens,

  Following is a sample content of a SOAP message that represents the response for the request of username token information:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
 xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
 <s:Body>
   <wsse:RequestSecurityTokenResponse>
    <wsse:RequestedSecurityToken>
      <wsse:UserNameToken
            <wsse:Username>peer2</wsse:Username>
            <wsse:Password>peer2</wsse:Password>
      </wsse:UserNameToken>
    </wsse:RequestedSecurityToken>
   </wsse:RequestSecurityTokenResponse>
 </s:Body>
</s:Envelope>
```

- validation of security tokens.

  Following is a sample content of a SOAP message that represents the validation request of username token information:

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

```
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
 xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
 <s:Body>
  <wsse:RequestSecurityToken>
   <wsse:TokenType>wsse:UserName</wsse:TokenType>
   <wsse:RequestType>wsse:ReqIssue</wsse:RequestType>
   <wsse:Base>
    <wsse:Reference URI="#peer2"/>
   </wsse:Base>
  </wsse:RequestSecurityToken>
 </s:Body>
</s:Envelope>
```

With the functionalities described above, security token service provides an interface for implementing security token management. Thus, the peers of this group behave as a trusted third party for the peers of PeerGroup-1 and PeerGroup-2.

CHAPTER IV

SECURE WEB SERVICE INVOCATION OVER JXTA FRAMEWORK

Within the scope of this thesis work, the implementation of peer-to-peer network and the communication among the peers of the peer groups decsribed in the previous chapter have been achieved by using the JXTA framework.

The JXTA framework requires the implementation of the JXTA protocol stack so that the peers conforming to these protocols become JXTA peers. The peers of PeerGroup-1, PeerGroup-2, and PeerGroup-3 imply to be JXTA peers with their implementations conforming to the protocols defined by JXTA framework.

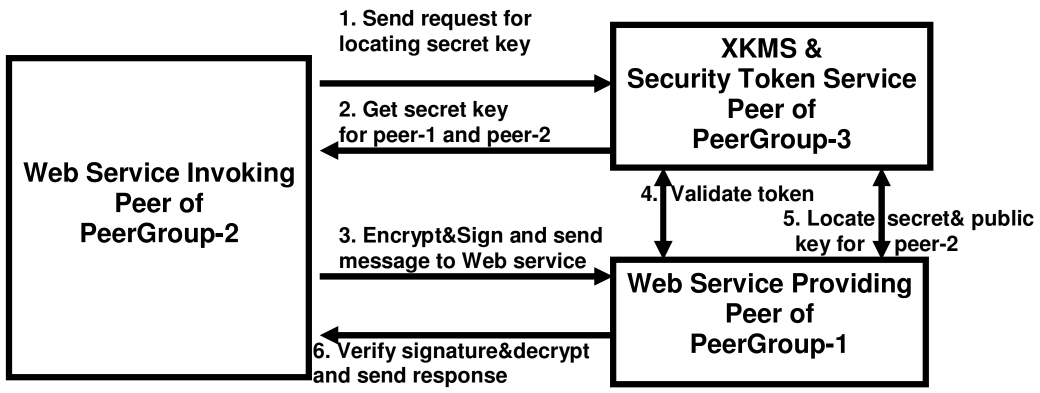
The JXTA framework facilitates the following operations to be exploited within the implementation of this work:

- authentication of each peer prior to operating to perform their functionalities, and this feature is exploited by the peers of PeerGroup-1, PeerGroup-2 and PeerGroup-3.
- creation of peer-to-peer network that is introduced by this work and is composed of the peers of PeerGroup-1, PeerGroup-2 and PeerGroup-3.
- bidirectional pipe implementation between the peers of PeerGroup-1, PeerGroup-2 and PeerGroup-3 in pairs, which provides the invocation of the Web services introduced within this work.

The following section provides the details of the developed prototype and presents an example walkthrough of the system introduced.

4.1. Design and Implementation of the System

Figure 4.1. denotes the execution flow of how secure Web service invocation is achieved in the peer-to-peer network introduced.



1. Send request for locating secret key
XKMS & Security Token Service Peer of PeerGroup-3
2. Get secret key for peer-1 and peer-2
Web Service Invoking Peer of PeerGroup-2
4. Validate token
5. Locate secret& public key for peer-2
3. Encrypt&Sign and send message to Web service
Web Service Providing Peer of PeerGroup-1
6. Verify signature&decrypt and send response

Figure 4.1. Architecture of the Peer-to-peer System

Considering the architecture above, the following operations are performed sequentially to achieve secure Web service invocation provided by the peer of PeerGroup-1:

1. the peer of PeerGroup-2 sends the request for the location of the secret key generated for the peer1-peer2 communication to the peer of PeerGroup-3.

2. the peer of PeerGroup-3 sends the generated secret key to the peer of PeerGroup-2.

3. the peer of PeerGroup-2 encrypts the constructed SOAP message with the secret key, adds the digital signature computed with its private key, adds the username-password security token, and sends the final SOAP message to the peer of PeerGroup-1.

4. the peer of PeerGroup-1 sends the request for the validation of username-passwords security token information belonging to the peer of PeerGroup-2 to the peer of PeerGroup-3.

5. the peer of PeerGroup-1 sends the request for the location of the public key belonging to the peer of PeerGroup-2 to the peer of PeerGroup-3, and sends the request for the location of the secret key generated for the peer1-peer2 communication to the peer of PeerGroup-3.

6. the peer of PeerGroup-1 first verifies the digital signature available in the SOAP message with the located public key, then decrypts the SOAP message with the located secret key.

The demonstrational implementation of the introduced system is realized by using the technologies of JXTA framework, WSDL, SOAP, and implementing the introduced system with J2SE (Java-2 Standard Edition).

The WSDL descriptions of the sample service to be invoked in a secure manner and the services introduced within this work with the formats of the SOAP messages related with these services are presented in Appendix-A and Appendix-B.

For the purpose of processing the WSDL files of the sample service provided by the peer of PeerGroup-2, processing the services related to the specifications of WS-Trust and XKMS, and the policy file holding the security assertions of the peer of PeerGroup-2 in the form of the specification WS-SecurityPolicy, the Xerces XML parser of Apache is used.

The implementation of the WS-Security specification used by the peers of PeerGroup-1 and the peers of PeerGroup-2 is achieved by exploiting and extending the open source implementation of XML Security specifications provided by the TrustGateway API (application programming interface) by VeriSign [23]. This API provides a programming interface to apply digital signature and encryption operations on the DOM (Document Object Model) representation of the XML documents conforming to the specifications of XML Encryption and XML Digital Signature. This API has been extended to provide a programming interface to apply the security token, digital signature and encryption constructs conforming to the specification of WS-Security, and to achieve symmetric secret key and asymmetric key pairs (public and private keys) generation. The security component of J2SE (Java-2 Standard Edition) has been used for the purposes of key generation. The mechanism of processing the security assertions stored in the WS-SecurityPolicy compliant files has also been provided in this extended programming interface.

The development of the graphical user interfaces for the protypes of the peers of PeerGroup-1 and PeerGroup-2 has been achieved by using Java Swing components.

For the purpose of exchanging the SOAP messages regarding the XKMS and security token services provided by the peer of PeerGroup-3, and regarding the sample service provided by the peer of PeerGroup-1, the JXTA framework has been used by its programming interface for bidirectional pipes.

4.2. Example Walkthrough

In order to demonstrate the execution of the system developed, three JXTA peers, each of which represents a peer of the peer groups introduced within this work, constitute the sample system. These three peers execute and construct the peer-to-peer network on the same system for demonstrational purpose.

All three peers perform the authentication process as denoted in Figure 4.2., which is facilitated by JXTA framework.

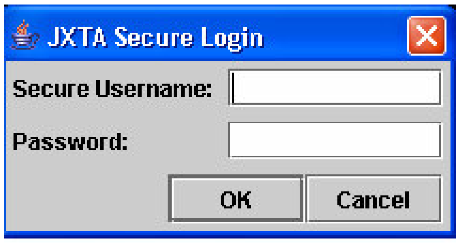
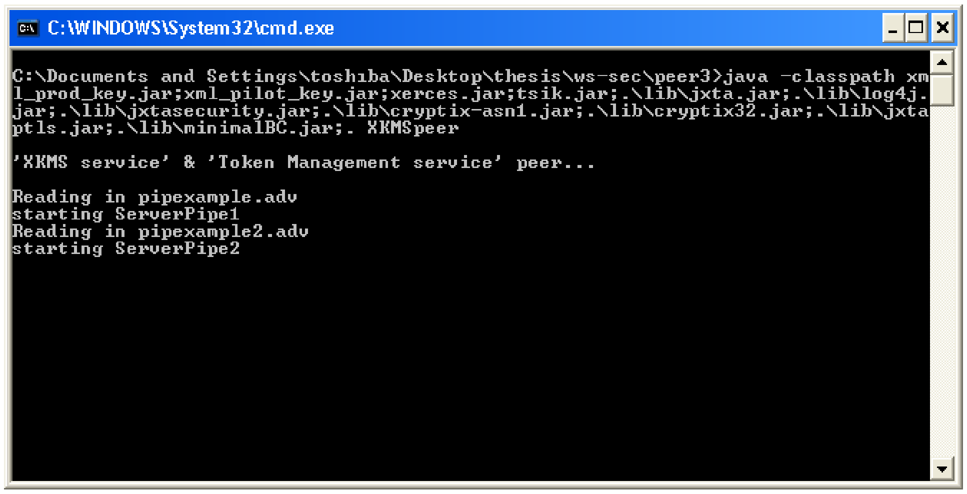


Figure 4.2. Authentication for Peer-to-peer System

The peer of PeerGroup-3 executes continuously in order to provide the XML Key Management service and security token service for the other two peers. Figure 4.3. denotes the execution of this peer.



```
C:\Documents and Settings\toshiba\Desktop\thesis\ws-sec\peer3>java -classpath xm
l_prod_key.jar;xml_pilot_key.jar;xerces.jar;tsik.jar;.\lib\jxta.jar;.\lib\log4j.
jar;.\lib\jxtasecurity.jar;.\lib\cryptix-asn1.jar;.\lib\cryptix32.jar;.\lib\jxta
ptls.jar;.\lib\minimalBC.jar;. XKMSpeer

'XKMS service' & 'Token Management service' peer...

Reading in pipexample.adv
starting ServerPipe1
Reading in pipexample2.adv
starting ServerPipe2
```

Figure 4.3. XML Key Management and Security Token Service Peer

This peer starts two pipe connections for the other two peers, which is facilitated by the JXTA framework. These two pipe connections are advertised to the corresponding peers with the following pipe advertisement structure of the JXTA framework:

```
<!DOCTYPE jxta:PipeAdvertisement>

<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
      <Id>
            urn:jxta:uuid-
59616261646162614A757874614D504725184FBC4E5D498AA0919F662
E40028B04
      </Id>
```

```
            <Type>
                  JxtaUnicast
            </Type>
            <Name>
                  PipeExample
            </Name>
      </jxta:PipeAdvertisement>
```

The peer of PeerGroup-2 provides the user interface, denoted in Figure 4.4., in order to construct the SOAP message for the invocation of the Web service that the peer of PeerGroup-1 provides.

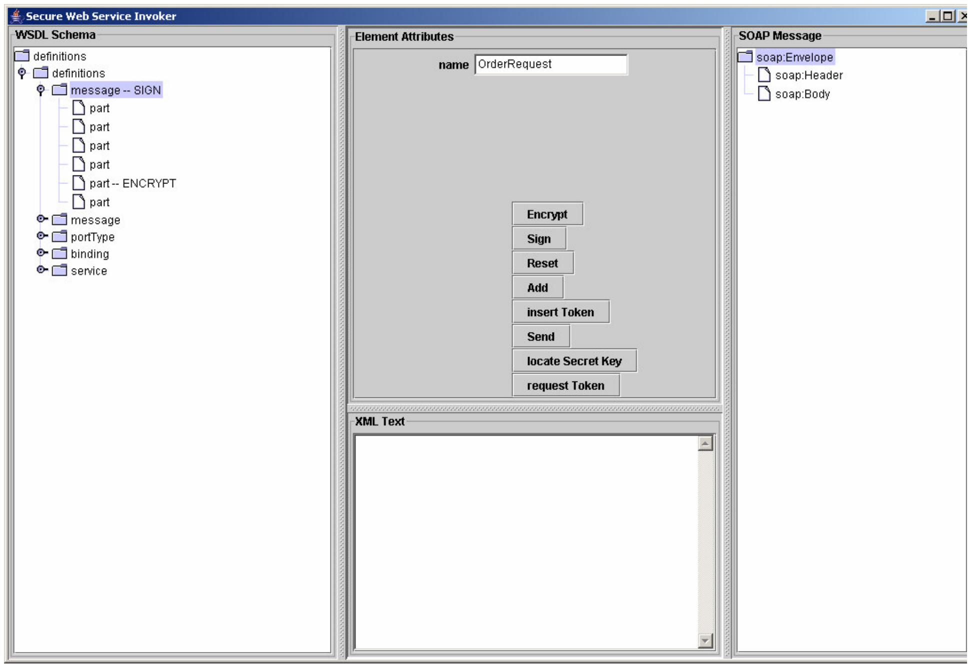
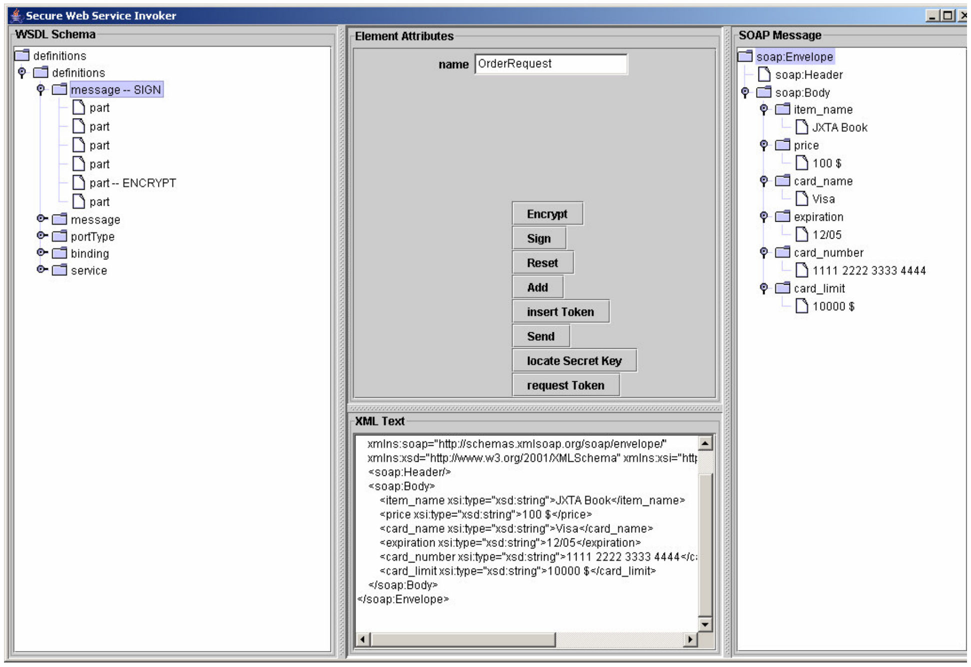


Figure 4.4. Secure Web Service Invoking Peer

The left part seen in Figure 4.4. is dedicated to the hierarchical view of the WSDL file of the sample Web service provided by the peer of PeerGroup-1, which reveals the granularity and the types of input parameter information. The sample Web service requires six input parameters denoted with “part” nodes on the WSDL view for an Order Request as seen in Figure 4.4.

Selecting each “part” node on the WSDL view and entering the required information conforming to the required parameter type information denoted in the middle part of Figure 4.4., the SOAP message is constructed on the right part of Figure 4.4. packed in the “soap:Body” element. Figure 4.5. denotes the constructed SOAP message:



```
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="htt
  <soap:Header/>
  <soap:Body>
    <item_name xsi:type="xsd:string">JXTA Book</item_name>
    <price xsi:type="xsd:string">100 $</price>
    <card_name xsi:type="xsd:string">Visa</card_name>
    <expiration xsi:type="xsd:string">12/05</expiration>
    <card_number xsi:type="xsd:string">1111 2222 3333 4444</c
    <card_limit xsi:type="xsd:string">10000 $</card_limit>
  </soap:Body>
</soap:Envelope>
```

Figure 4.5. Constructed SOAP Message

The constructed SOAP message is:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Header/>
  <soap:Body>
    <item_name xsi:type="xsd:string">JXTA Book</item_name>
    <price xsi:type="xsd:string">1000 $</price>
    <card_name xsi:type="xsd:string">Visa</card_name>
    <expiration xsi:type="xsd:string">10/04</expiration>
    <card_number xsi:type="xsd:string">1111 2222 3333 4444</card_number>
    <card_limit xsi:type="xsd:string">10000 $</card_limit>
  </soap:Body>
</soap:Envelope>
```

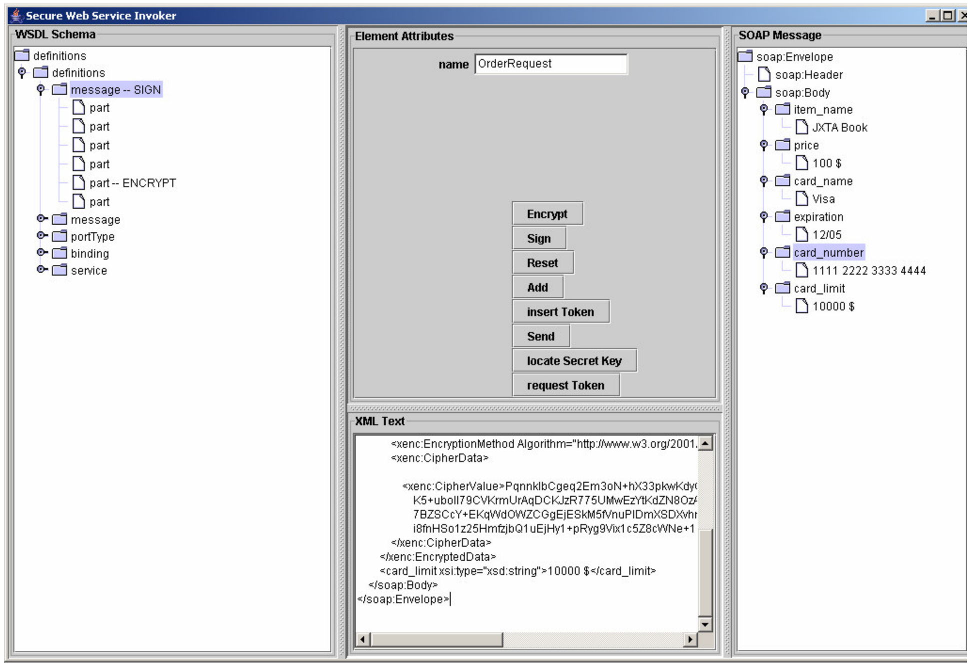
In order to conceal the credit card number information, which is one of the input parameters of the “Order Request” service, firstly, the secret key information is requested from the peer of PeerGroup-3, which is running continuously to respond. The secret key request message is:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"

xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
>
 <s:Body>
  <RequestSecretKey>
   <Party1>peer1</Party1>
   <Party2>peer2</Party2>
  </RequestSecretKey>
 </s:Body>
</s:Envelope>
```

The secret key response message created by the peer of PeerGroup-3 is:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"

xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"

xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
>
 <s:Body>
  <ResponseSecretKey>
   <Party1>peer1</Party1>
   <Party2>peer2</Party2>
      <SecretKey>Rz_Õ,|pÖkµÈsÜµñ¿‰ ?ÂßŠm^</SecretKey>
  </ResponseSecretKey>
 </s:Body>
</s:Envelope>
```

Selecting the "card_number" node on the SOAP message view, the constructed SOAP message is encrypted on the "card_number" element with the received secret key information after checking the security policy assertions stored in the WS-SecurityPolicy compliant file if this action is allowed by the service provider. Figure 4.6. denotes the encryption applied to the constructed SOAP message to conceal the credit card number information:



```
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001.
        <xenc:CipherData>

          <xenc:CipherValue>PqnnklbCgeq2Em3oN+hX33pkwKdy
            K5+uboll79CVKrmUrAqDCKJzR775UMwEzYtKdZN8Oz
            7BZSCcY+EKqWdOWZCGgEjESkM5fVnuPIDmXSDXvh
            i8fnHSo1z25HmfzjbQ1uEjHy1+pRyg9Vix1c5Z8cWNe+1
        </xenc:CipherData>
      </xenc:EncryptedData>
      <card_limit xsi:type="xsd:string">10000 $</card_limit>
   </soap:Body>
</soap:Envelope>
```

Figure 4.6. Resultant SOAP Message after Encryption

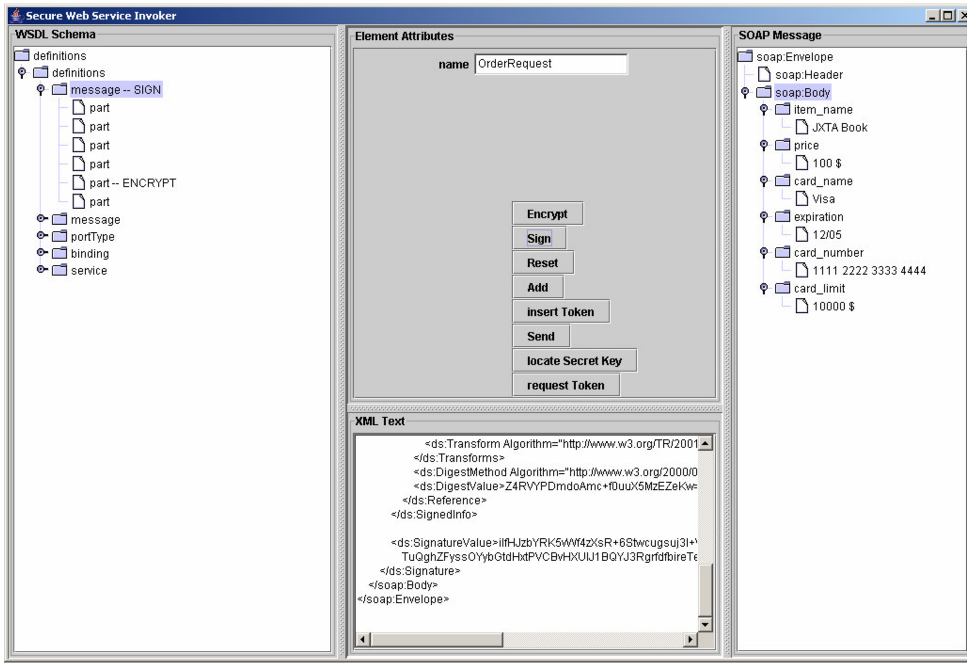
The following is the content of the security policy file for the Web service provided by the peer of PeerGroup-1:

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
      <wsse:SecurityToken wsp:Usage="wsp:Rejected">
            <wsse:TokenType>wsse:Kerberosv5ST</wsse:TokenType>
      </wsse:SecurityToken>
      <wsse:SecurityToken wsp:Usage="wsp:Required">
            <wsse:TokenType>wsse:UsernameToken</wsse:TokenType>
      </wsse:SecurityToken>
      <wsse:Integrity wsp:Usage="wsp:Required">
            <wsse:Algorithm Type="wsse:AlgSignature" />
            <MessageParts> /soap:Body </MessageParts>
      </wsse:Integrity>
      <wsse:Confidentiality wsp:Usage="wsp:Required">
            <wsse:Algorithm Type="wsse:AlgEncryption" />
            <MessageParts> /soap:Body/card_number
            </MessageParts>
      </wsse:Confidentiality>
</wsp:Policy>
```

The resultant SOAP message after encryption is:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
   soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soap:Header/>
   <soap:Body>
      <item_name xsi:type="xsd:string">JXTA Book</item_name>
      <price xsi:type="xsd:string">1000 $</price>
      <card_name xsi:type="xsd:string">Visa</card_name>
      <expiration xsi:type="xsd:string">10/04</expiration>
      <xenc:EncryptedData
         Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
         <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/>
         <xenc:CipherData>


<xenc:CipherValue>wLdVyjsa1XWta8OIFdOd05PIFfBQINZG500jibVHO+XwBqLExe4WN
cQPZVLJjXTljRoItvsm

JaFzOOf7fm3NvZ+M8UATgo6vPLkE88ebrIQv0a3YPNHajyFykIrJOgfDp++9QFRcdOyHU
mS+

3n1w2xhofrw4GGKW+/jIYB8AkobpofcUX9OMv/ZuLxDurmxjDfR4q7ndmRg8XRVPBUT
NTSNh

kcxZyJVMKDRUKy/CLhaElmI5Jn2IFkwh2orDgfZJUyo5cYS+GFtjcZVz7cPOAXdKzIz+6U
76 jIIdfWo4usD59ldFil3/Kg==</xenc:CipherValue>
         </xenc:CipherData>
      </xenc:EncryptedData>
      <card_limit xsi:type="xsd:string">10000 $</card_limit>
   </soap:Body>
</soap:Envelope>
```

In order to add digital signature to the SOAP message for enabling message integrity and non-repudiation of this peer, selecting the “soap:Body” node on the SOAP message view, the constructed SOAP message is extended with the digital signature obtained on the “soap:Body” element with the private key information of this peer after checking the security policy assertions stored in the WS-SecurityPolicy compliant file if this action is allowed by the service provider. Figure 4.7. denotes the SOAP message after applying the digital signature to the whole SOAP message content to avoid any modification:



```
definitions
  definitions
    message -- SIGN
      part
      part
      part
      part
      part -- ENCRYPT
      part
    message
    portType
    binding
    service
```

name: OrderRequest

```
                <ds:Transform Algorithm="http://www.w3.org/TR/2001
              </ds:Transforms>
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/0
              <ds:DigestValue>Z4RVYPDmdoAmc+f0uuX5MzEZeKw=
            </ds:Reference>
          </ds:SignedInfo>

          <ds:SignatureValue>ilfHJzbYRK5WVf4zXsR+6Stwcugsuj3I+
            TuQghZFyssOYybGtdHxtPVCBvHXUlJ1BQYJ3RgrfdfbireT
        </ds:Signature>
    </soap:Body>
</soap:Envelope>
```

```
soap:Envelope
  soap:Header
  soap:Body
    item_name
      JXTA Book
    price
      100 $
    card_name
      Visa
    expiration
      12/05
    card_number
      1111 2222 3333 4444
    card_limit
      10000 $
```

Figure 4.7. Resultant SOAP Message after Adding Digital Signature

The resultant SOAP message after adding digital signature is:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
   soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soap:Header/>
   <soap:Body>
      <item_name xsi:type="xsd:string">JXTA Book</item_name>
      <price xsi:type="xsd:string">1000 $</price>
      <card_name xsi:type="xsd:string">Visa</card_name>
      <expiration xsi:type="xsd:string">10/04</expiration>
      <xenc:EncryptedData
         Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
         <xenc:EncryptionMethod     Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/>
         <xenc:CipherData>

<xenc:CipherValue>wLdVyjsa1XWta8OIFdOd05PIFfBQINZG500jibVHO+XwBqLExe4WN
cQPZVLJjXTljRoItvsm
JaFzOOf7fm3NvZ+M8UATgo6vPLkE88ebrIQv0a3YPNHajyFykIrJOgfDp++9QFRcdOyHU
mS+
3n1w2xhofrw4GGKW+/jIYB8AkobpofcUX9OMv/ZuLxDurmxjDfR4q7ndmRg8XRVPBUT
NTSNh
```

kcxZyJVMKDRUKy/CLhaElmI5Jn2IFkwh2orDgfZJUyo5cYS+GFtjcZVz7cPOAXdKzIz+6U
76 jIIdfWo4usD59ldFil3/Kg==</xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedData>
      <card_limit xsi:type="xsd:string">10000 $</card_limit>
      <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"/>
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
          <ds:Reference URI="">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
              <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <ds:DigestValue>7u8LlBJH4v/d2PnCZ4K+UM/IOAk=</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>

<ds:SignatureValue>HiLFR9IJL4idhw1ddKLG9ZseUnmrNqgKYy3ubVAhB3qJNF+s8qwNq
+uU7FnlCk+Dp+U/CFpV

WakpfuOJntTSfiFBLziPCpRdFr/T54EbkRSbsMm5kvAhA96aiajooLJZnwmq4gADo1GGarus
gQlI4U9DPs3cVEhqhhLLj+OqSQA=</ds:SignatureValue>
      </ds:Signature>
    </soap:Body>
</soap:Envelope>

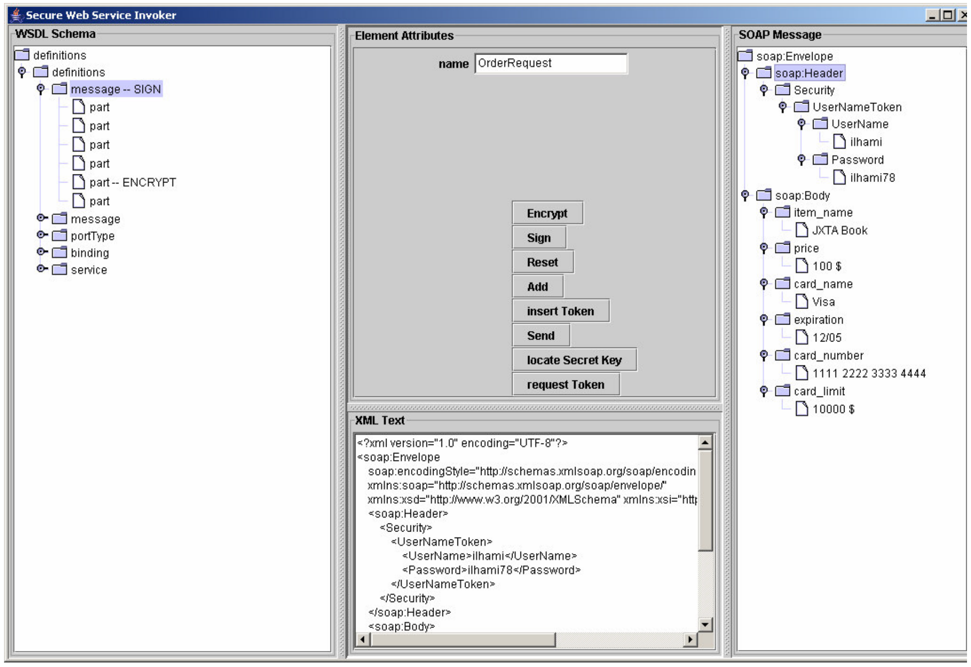
To provide the security token information in the SOAP message, firstly, the username-password token information is requested from the peer of PeerGroup-3, which is running continuously to respond. The request message for the username-password token information is:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
 xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
 <s:Body>
  <wsse:RequestSecurityToken>
   <wsse:TokenType>wsse:UserName</wsse:TokenType>
   <wsse:RequestType>wsse:ReqIssue</wsse:RequestType>
   <wsse:Base>
    <wsse:Reference URI="#peer2"/>
   </wsse:Base>
  </wsse:RequestSecurityToken>
 </s:Body>
</s:Envelope>
```

The token information response message created by the peer of PeerGroup-3 is:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope
 xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/12/secext"
 xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility">
 <s:Body>
   <wsse:RequestSecurityTokenResponse>
    <wsse:RequestedSecurityToken>
      <wsse:UserNameToken
      <wsse:Username>ilhami</wsse:Username>
      <wsse:Password>ilhami78</wsse:Password>
      </wsse:UserNameToken>
    </wsse:RequestedSecurityToken>
   </wsse:RequestSecurityTokenResponse>
 </s:Body>
</s:Envelope>
```

The constructed SOAP message is extended with the received secret key information by adding this information on the “soap:Header” element of the SOAP message view after checking the security policy assertions stored in the WS-SecurityPolicy compliant file if this action is allowed by the service provider. Figure 4.8. denotes the resultant SOAP message content after inserting security token information which is achieved by username-password information mechanism:



name OrderRequest

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encodin
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http
  <soap:Header>
    <Security>
      <UserNameToken>
        <UserName>ilhami</UserName>
        <Password>ilhami78</Password>
      </UserNameToken>
    </Security>
  </soap:Header>
  <soap:Body>
```

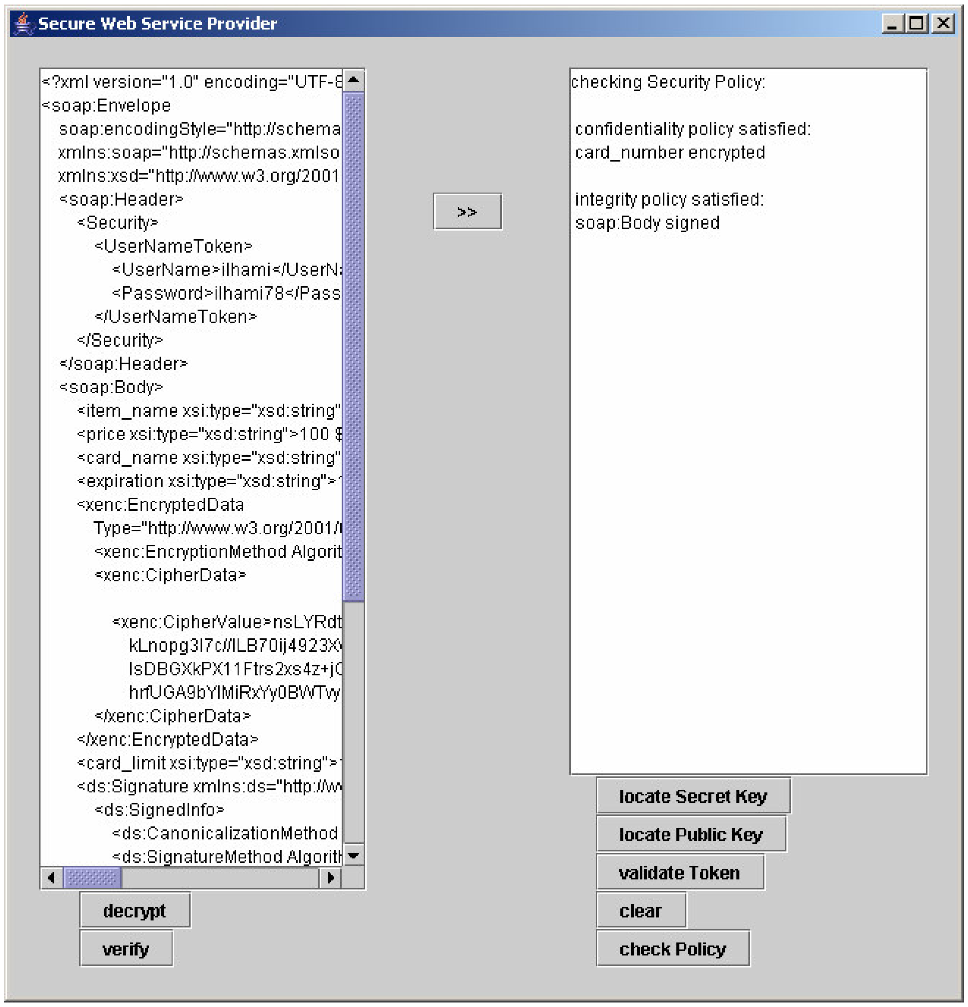
Figure 4.8. Resultant SOAP Message after Adding Username-Password Token Information

The final SOAP message is:

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Header>
    <Security>
      <UserNameToken>
        <UserName>ilhami</UserName>
        <Password>ilhami78</Password>
      </UserNameToken>
    </Security>
  </soap:Header>
  <soap:Body>
    <item_name xsi:type="xsd:string">JXTA Book</item_name>
    <price xsi:type="xsd:string">1000 $</price>
    <card_name xsi:type="xsd:string">Visa</card_name>
    <expiration xsi:type="xsd:string">10/04</expiration>
    <xenc:EncryptedData
      Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      <xenc:EncryptionMethod    Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/>
```

```
      <xenc:CipherData>
<xenc:CipherValue>YstBEwHRXKXx2VboMrzwENXPqAJgA52by9dxVkJ5q5FlTNf0QPm6
e0dYW1QbY7LfK9danRQf
5j4xyXG7YSPKyWCycJlzYw+J1YOVKX7xxSCb2KrvMsVPenHO+bZwPDRmYj8Ra3IcpX
TkcoGw
mQ+Pal6UBuYcB3cRuM5u7WL6Hvsj50sXJ0tlpzzpDgWBW6RpG1i9P51905ECmTZhoqyQ
rZ8m
XtXxfwbZN4iQRWdSVlZRjLboKACVY5PQMktD8V9JspCHXR1ZRBIHvKdockhnix2+/LA
h0tOn LzpHf9bh6aAn9jmkI38U/A==</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
    <card_limit xsi:type="xsd:string">10000 $</card_limit>
    <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"/>
        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="">
          <ds:Transforms>
            <ds:Transform      Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature"/>
            <ds:Transform           Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-
20010315"/>
          </ds:Transforms>
          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
          <ds:DigestValue>1L365M10r20brde/TTozz/YAMTc=</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
<ds:SignatureValue>RFBtZs6QbJvlj0nQK7urx8XKTfoKL+0/I4F3HEBYufqoFHuT1SPjpQP
AqnXiui2wpjbXkOiO
GkjY3M1FsiFW6v3Bdl9ejNEaryhsZimJSS3Tmlu2U0ZI8JQL4pXYUsZrx78Wv5zRWH2Prv
XW HCsGub1JsB2LVGGmikDHswKWxRI=</ds:SignatureValue>
    </ds:Signature>
  </soap:Body>
</soap:Envelope>
```

The peer of PeerGroup-1 provides the user interface, denoted in Figure 4.9., in order to process the security information available in the SOAP message received from the peer of PeerGroup-2 by initially checking the obedience of the security policies.



```
<?xml version="1.0" encoding="UTF-8
<soap:Envelope
  soap:encodingStyle="http://schema
  xmlns:soap="http://schemas.xmlso
  xmlns:xsd="http://www.w3.org/2001
  <soap:Header>
    <Security>
      <UserNameToken>
        <UserName>ilhami</UserN
        <Password>ilhami78</Pass
      </UserNameToken>
    </Security>
  </soap:Header>
  <soap:Body>
    <item_name xsi:type="xsd:string"
    <price xsi:type="xsd:string">100 $
    <card_name xsi:type="xsd:string"
    <expiration xsi:type="xsd:string">
    <xenc:EncryptedData
      Type="http://www.w3.org/2001/
      <xenc:EncryptionMethod Algorit
      <xenc:CipherData>

        <xenc:CipherValue>nsLYRdt
          kLnopg3I7c//ILB70ij4923X
          IsDBGXkPX11Ftrs2xs4z+jC
          hrfUGA9bYIMiRxYy0BWTy
      </xenc:CipherData>
    </xenc:EncryptedData>
    <card_limit xsi:type="xsd:string">
    <ds:Signature xmlns:ds="http://w
      <ds:SignedInfo>
        <ds:CanonicalizationMethod
        <ds:SignatureMethod Algorit
```

```
checking Security Policy:

confidentiality policy satisfied:
card_number encrypted

integrity policy satisfied:
soap:Body signed
```

Figure 4.9. Received Secure SOAP Message

Figure 4.10. denotes the process for verifying the digital signature applied to the received message by the peer of PeerGroup-2.

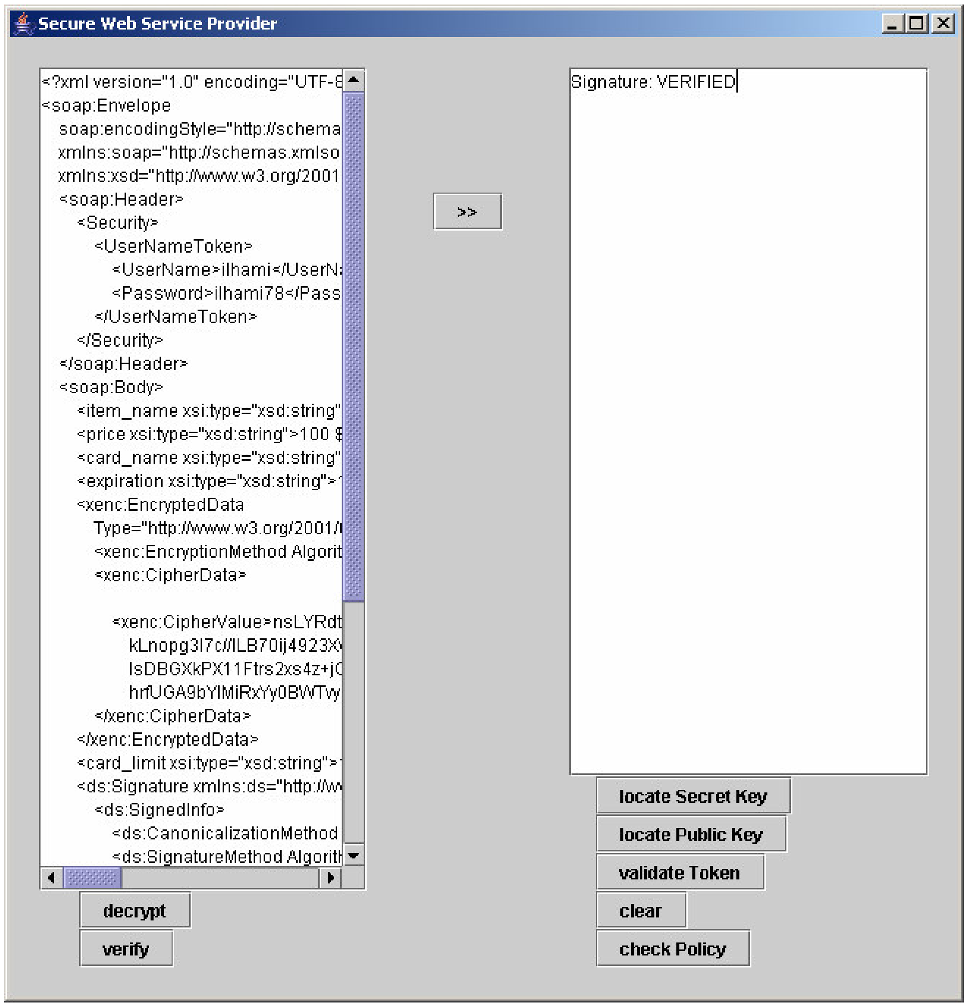
Prior to the verification of the digital signature, the public key information for the peer of PeerGroup-2 is located by requesting this key content from the peer of PeerGroup-3. The public key information location request message is:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
      <k:Locate xmlns:k="http://www.xkms.org/schema/xkms-2001-
01-20">
```

```
            <k:TransactionID>c41696b0-1f14-11d6-b840-
      3beb2501bc66
            </k:TransactionID>
            <k:Query>
                  <d:KeyInfo
            xmlns:d="http://www.w3.org/2000/09/xmldsig#">
                  <d:KeyName>peer2</d:KeyName>
                  </d:KeyInfo>
            </k:Query>
            <k:Respond>
                  <k:string>KeyName</k:string>
                  <k:string>KeyValue</k:string>
            </k:Respond>
      </k:Locate>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The public key information response message created by the peer of PeerGroup-3 is:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">"
<SOAP-ENV:Body>
      <k:LocateResult
xmlns:k=\"http://www.xkms.org/schema/xkms-2001-01-20\">
            <k:TransactionID>c41696b0-1f14-11d6-b840-
      3beb2501bc66</k:TransactionID>
            <Result>Success</Result>
            <Answer>
            <KeyInfo
xmlns="http://www.w3.org/2000/09/xmldsig#">
                  <KeyName>peer2</KeyName>
                  <RetrievalMethod
Type="http://service.xmltrustcenter.org/XKMS"
URI="XKMSpeer:xkms/Acceptor.nano"/>
                  <d:KeyValue
xmlns:d="http://www.w3.org/2000/09/xmldsig#">
                        <d:RSAKeyValue>
<d:Modulus>zYQky4msu0MYMM7d3npKhaoRdy9YkVcYAquZ7F2dt3GCyRN4dhq3
v33cfOu8JRCw2DVTctXSMRx4drCA0JQH6wNXekViuvMrS/OS920esgsL7q5zO0t
5jHw2FVT2Fci3luqJDE4Pg/9hFbQs1JTnmZfRskhjJtKpb8Op5ahcmtc=
</d:Modulus>
<d:Exponent xmlns:d="http://www.w3.org/2000/09/xmldsig#">AQAB
</d:Exponent>
                        </d:RSAKeyValue>
                  </d:KeyValue>
            </KeyInfo>
      </LocateResult>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```



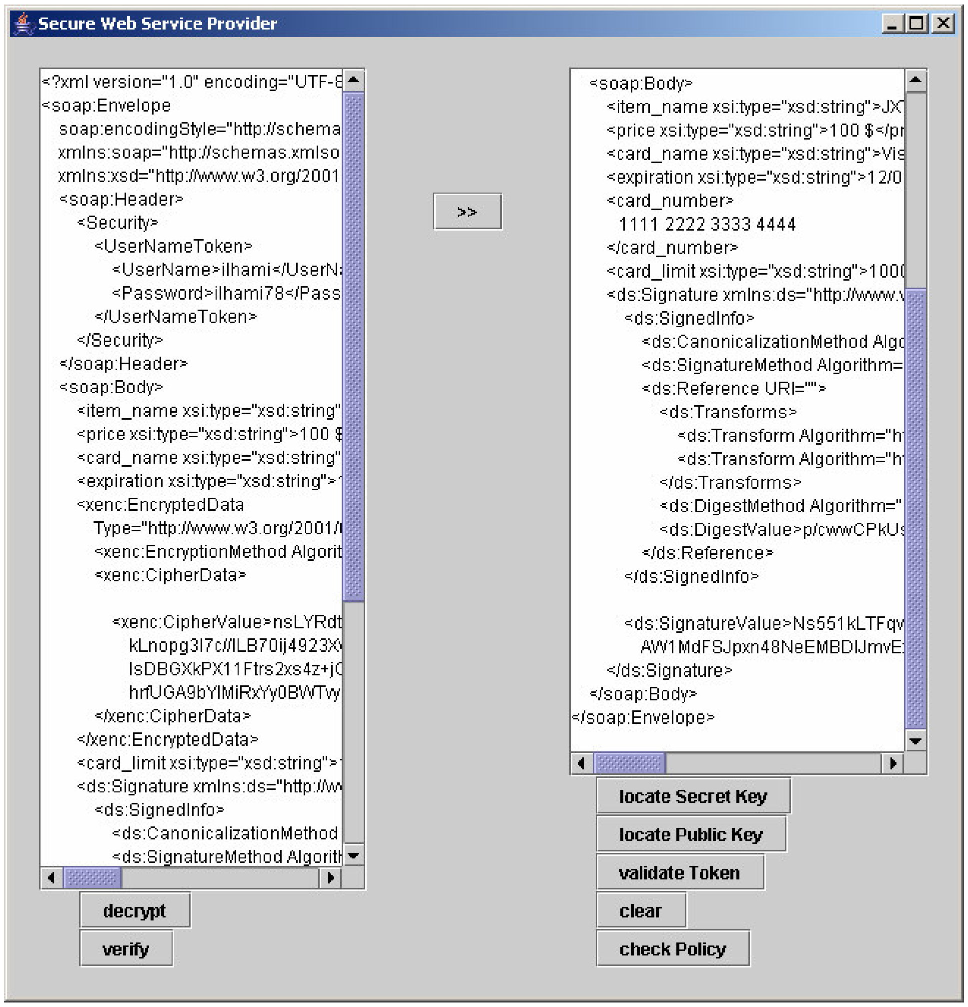
```
<?xml version="1.0" encoding="UTF-8
<soap:Envelope
  soap:encodingStyle="http://schema
  xmlns:soap="http://schemas.xmlso
  xmlns:xsd="http://www.w3.org/2001
  <soap:Header>
    <Security>
      <UserNameToken>
        <UserName>ilhami</UserN
        <Password>ilhami78</Pass
      </UserNameToken>
    </Security>
  </soap:Header>
  <soap:Body>
    <item_name xsi:type="xsd:string"
    <price xsi:type="xsd:string">100 $
    <card_name xsi:type="xsd:string"
    <expiration xsi:type="xsd:string">
    <xenc:EncryptedData
      Type="http://www.w3.org/2001/
      <xenc:EncryptionMethod Algorit
      <xenc:CipherData>

        <xenc:CipherValue>nsLYRdt
          kLnopg3l7c//ILB70ij4923X
          IsDBGXkPX11Ftrs2xs4z+jC
          hrfUGA9bYIMiRxYy0BWTy
      </xenc:CipherData>
    </xenc:EncryptedData>
    <card_limit xsi:type="xsd:string">
    <ds:Signature xmlns:ds="http://w
      <ds:SignedInfo>
        <ds:CanonicalizationMethod
        <ds:SignatureMethod Algorit
```

```
Signature: VERIFIED
```

Figure 4.10. Digital Signature Verification

Figure 4.11. denotes the process for decrypting the encrypted SOAP message element that has been applied to the credit card number information in the received message by the peer of PeerGroup-2.

Prior to decryption, the secret key information, that has been created by the peer of PeerGroup-3 for the connection established between the peer of PeerGroup-1 and the peer of PeerGroup-2, is located by requesting this key content from the peer of PeerGroup-3.



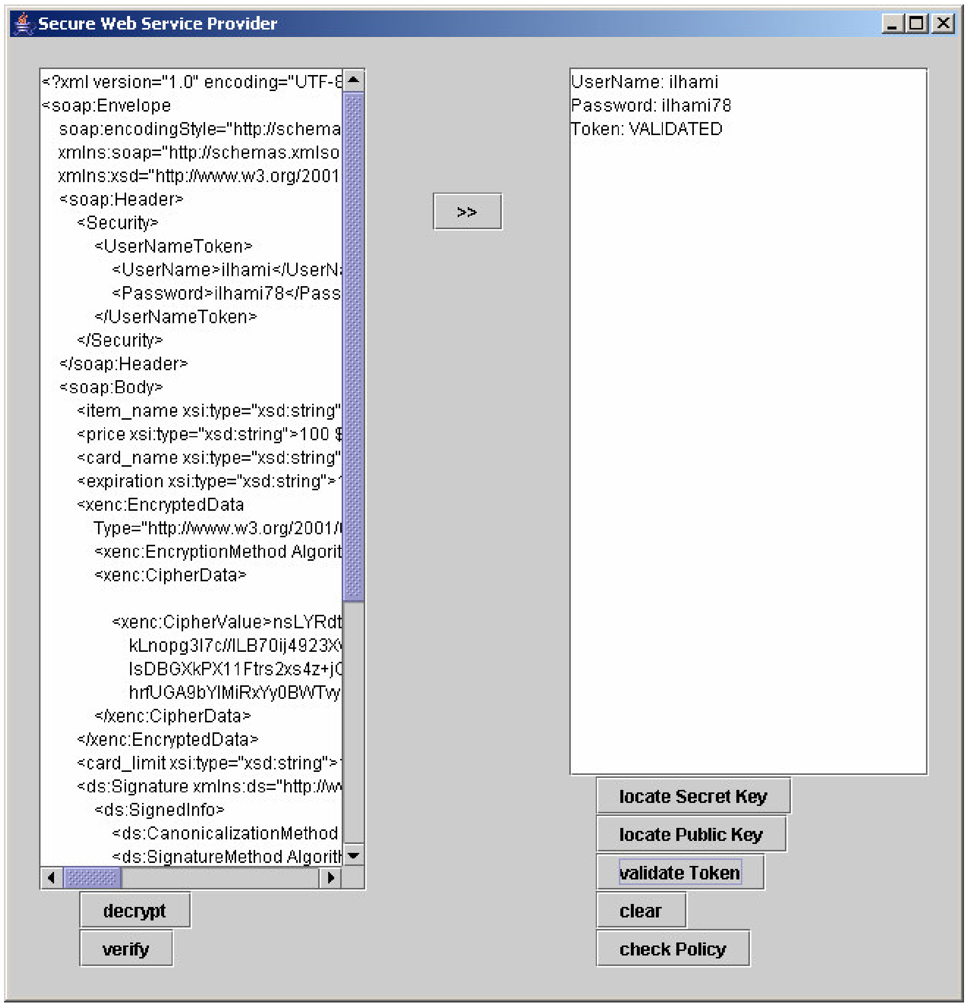
```
<?xml version="1.0" encoding="UTF-8
<soap:Envelope
  soap:encodingStyle="http://schema
  xmlns:soap="http://schemas.xmlso
  xmlns:xsd="http://www.w3.org/2001
  <soap:Header>
    <Security>
      <UserNameToken>
        <UserName>ilhami</UserN
        <Password>ilhami78</Pass
      </UserNameToken>
    </Security>
  </soap:Header>
  <soap:Body>
    <item_name xsi:type="xsd:string"
    <price xsi:type="xsd:string">100 $
    <card_name xsi:type="xsd:string"
    <expiration xsi:type="xsd:string">1
    <xenc:EncryptedData
      Type="http://www.w3.org/2001/
      <xenc:EncryptionMethod Algorit
      <xenc:CipherData>

        <xenc:CipherValue>nsLYRdt
          kLnopg3I7c//ILB70ij4923X
          IsDBGXkPX11Ftrs2xs4z+jC
          hrfUGA9bYIMiRxYy0BWTy
      </xenc:CipherData>
    </xenc:EncryptedData>
    <card_limit xsi:type="xsd:string">1
    <ds:Signature xmlns:ds="http://w
      <ds:SignedInfo>
        <ds:CanonicalizationMethod
        <ds:SignatureMethod Algorith
```

```
  <soap:Body>
    <item_name xsi:type="xsd:string">JX
    <price xsi:type="xsd:string">100 $</pr
    <card_name xsi:type="xsd:string">Vis
    <expiration xsi:type="xsd:string">12/0
    <card_number>
      1111 2222 3333 4444
    </card_number>
    <card_limit xsi:type="xsd:string">1000
    <ds:Signature xmlns:ds="http://www.v
      <ds:SignedInfo>
        <ds:CanonicalizationMethod Algo
        <ds:SignatureMethod Algorithm=
        <ds:Reference URI="">
          <ds:Transforms>
            <ds:Transform Algorithm="ht
            <ds:Transform Algorithm="ht
          </ds:Transforms>
          <ds:DigestMethod Algorithm="
          <ds:DigestValue>p/cwwCPkUs
        </ds:Reference>
      </ds:SignedInfo>

      <ds:SignatureValue>Ns551kLTFqv
        AW1MdFSJpxn48NeEMBDIJmvE
    </ds:Signature>
  </soap:Body>
</soap:Envelope>
```

Figure 4.11. Resultant SOAP Message after Decryption

Figure 4.12. denotes the process for validating the username security token information inserted to the received message by the peer of PeerGroup-2.

Prior to the validation of the security token, the username token information for the peer of PeerGroup-2 is located by requesting from the peer of PeerGroup-3.



```
<?xml version="1.0" encoding="UTF-8
<soap:Envelope
  soap:encodingStyle="http://schema
  xmlns:soap="http://schemas.xmlso
  xmlns:xsd="http://www.w3.org/2001
  <soap:Header>
    <Security>
      <UserNameToken>
        <UserName>ilhami</UserN
        <Password>ilhami78</Pass
      </UserNameToken>
    </Security>
  </soap:Header>
  <soap:Body>
    <item_name xsi:type="xsd:string"
    <price xsi:type="xsd:string">100 $
    <card_name xsi:type="xsd:string"
    <expiration xsi:type="xsd:string">1
    <xenc:EncryptedData
      Type="http://www.w3.org/2001/
      <xenc:EncryptionMethod Algorit
      <xenc:CipherData>

        <xenc:CipherValue>nsLYRdt
          kLnopg3l7c//ILB70ij4923X
          IsDBGXkPX11Ftrs2xs4z+jC
          hrfUGA9bYlMiRxYy0BWTy
      </xenc:CipherData>
    </xenc:EncryptedData>
    <card_limit xsi:type="xsd:string">
    <ds:Signature xmlns:ds="http://w
      <ds:SignedInfo>
        <ds:CanonicalizationMethod
        <ds:SignatureMethod Algorit
```

```
UserName: ilhami
Password: ilhami78
Token: VALIDATED
```

Figure 4.12. Security Token Validation

CHAPTER V

THE SATINE PROJECT INTEGRATION

The thesis work is realized as a part of the SATINE project which is funded by the European Commission [24, 25, 26, 27, 28].

The SATINE project is aiming to develop a semantic-based interoperability framework for the tourism industry. The framework provides tools and mechanisms for publishing, discovering and invoking Web services through their semantics in peer-to-peer networks, thus exploiting the synergies between these two technologies. By means of the SATINE tools and infrastructure, tourism companies, such as hotel chains, rent-a-car agencies and airline companies, are able to:

- wrap their applications with Web services,
- enrich those services with semantic descriptions,
- publish them on the P2P network.

Service requestors, such as travel agencies, are able to discover services based on their semantics, invoke them, combine simple travel services to complex ones.

In order to be compliant with industry standards the Open Travel Alliance (OTA) specifications are considered for the messages exchanged between the trading partners. However OTA compliance is not enforced. Rather the automated mapping of messages and functionalities are supported through the use of appropriate ontologies. The infrastructure to be developed especially addresses the difficulties of smaller companies in announcing their services.

The SATINE architecture provides a framework which is based on P2P networks integrated with Service Registries. The architecture shown in Figure 5.1 enables searching, publishing, and invoking travel Web services in a distributed environment.

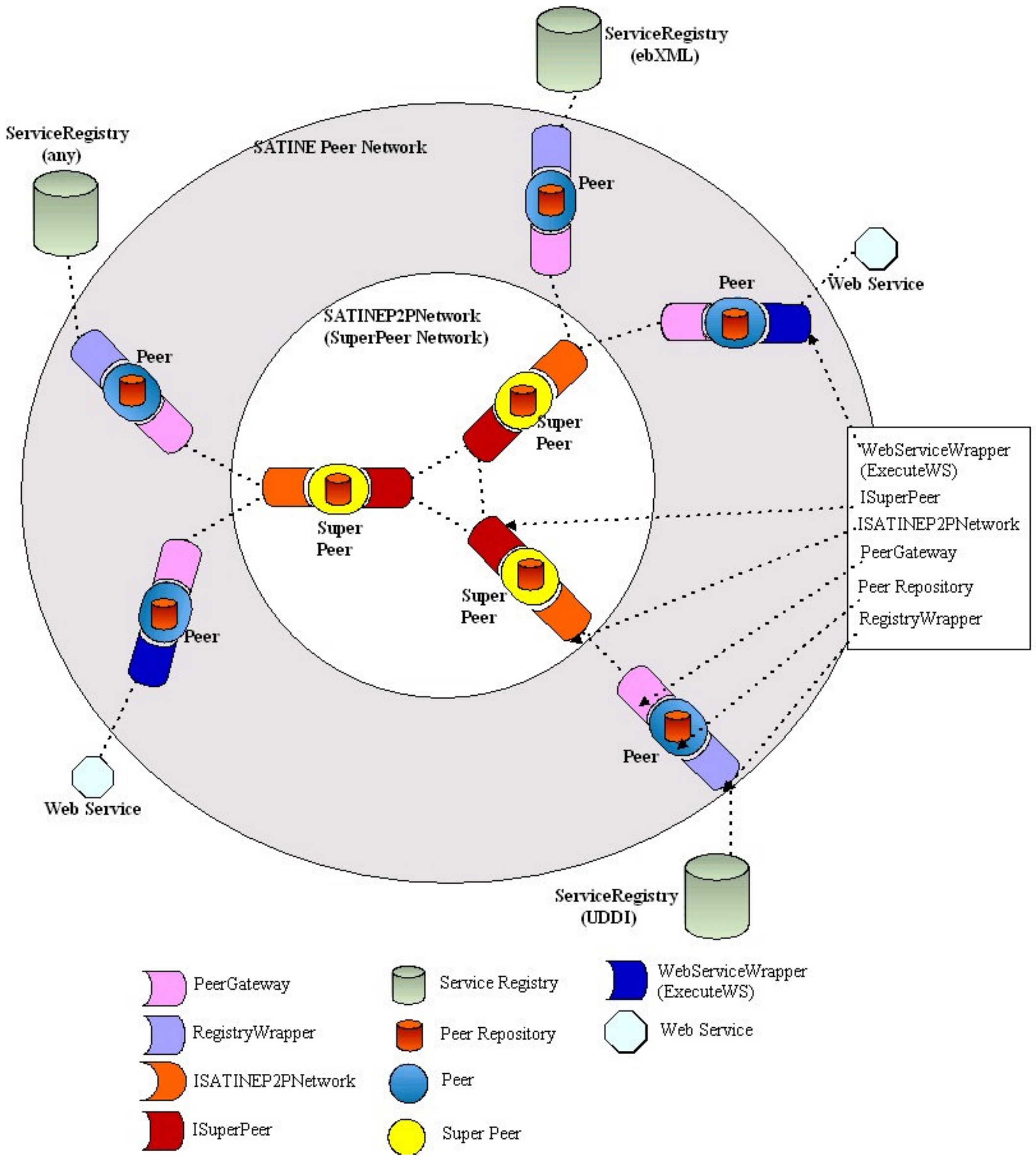


Figure 5.1. General Architecture

The network consists of three layers. The core layer is the super peer network which is named as SATINE P2P Network. Super peers are responsible for keeping the semantic routing tables and routing the messages which may be queries, advertisements, Web Service invocation/result messages and some other entities in the network. Super peers are connected each other via Super Peer interface. The second layer is the peer network. Peers represent the users of the SATINE architecture. This layer includes applications to help users in integrating their business to SATINE, get connected with Super Peer Network and to advertise and search services in SATINE.

Each peer is a travel service provider or requestor. Peers are connected to the lower layer which is super peer network, with PeerGateway interface. Every ingoing or outgoing function from/to peer is realized with use of this interface. Peers are able to wrap some Service Registries or Web Services and inform the remaining of the network from them.

Applications to wrap a Registry or a Web Service are included in the Peer Network Layer. The set of wrapped sources which are the Registries and Web Services defines the outmost layer. The Web Services may be already existing Web Services or some Web Service constructed with the tools provided by SATINE. Via the tools that will be provided it will be possible to wrap existing resources such as databases as Web Services.

Each peer in the system, including super peers, has a central peer repository in order to keep the temporary or persistent information for that peer. The information about the wrapped sources, semantic information about the peer such as the ontologies, and ontology mapping definitions, semantic routing tables are all examples for the set of information to be kept in central peer repositories. Peer repositories are totally different than business Service Registries/Repositories. Peer repositories are ordinary databases where information specific to a peer is kept in tables. Figure 5.2 shows the deployment diagram for this architecture.

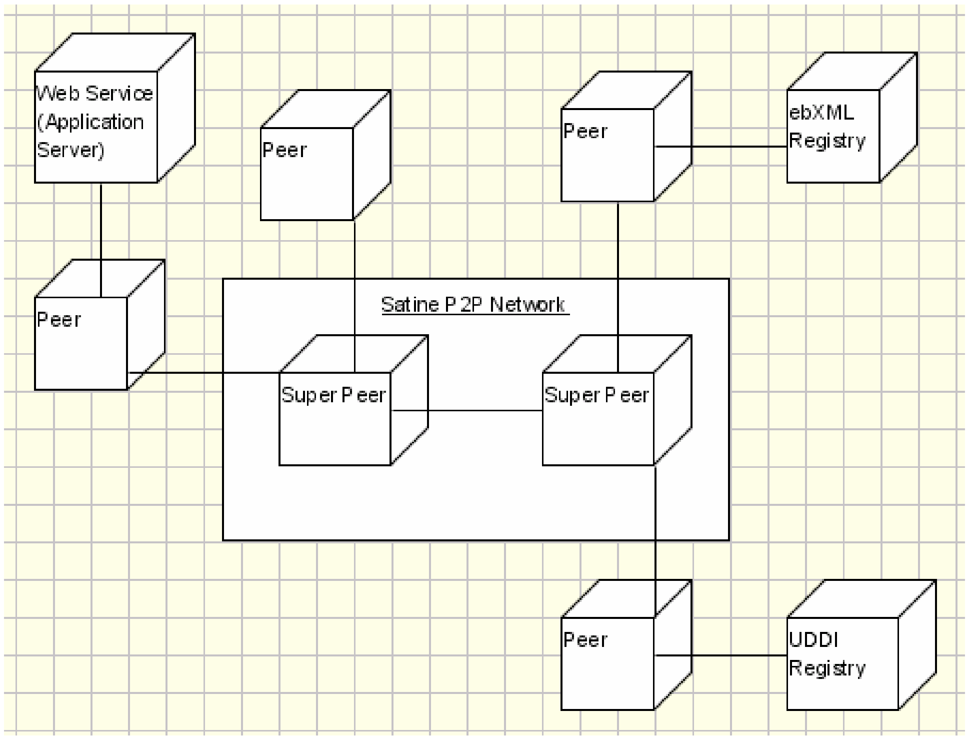


Figure 5.2. General Architecture Deployment Diagram

The Web Services and Registries in SATINE are advertised semantically to the network when wrapped by a peer in the system. As the ontologies are the core components for a semantic based architecture, the advertisements of Web Services or Registries are annotated with ontologies. The ontologies are previously deployed to the system. Applications in Peer Network Layer create the advertisements and pass the advertisements to the Super Peer Network. Super Peers update their semantic routing tables using these advertisements. SATINE supports advertisements and queries in more than one ontologies.

Instead of a single global ontology, it supports more than one global ontologies in the system, and provides way to map and translate semantically overlapping entities between these ontologies. Tools to design mapping and applications to translate between ontology instances are provided in Peers. Translation mechanism provides translation services to whole SATINE platform, which may be used by Peers or Super Peers. The system also provides the evolution of the ontologies via again mappings and translations.

5.1. SATINE Network Security

As this thesis work is realized as a part of the SATINE project, the architecture introduced with this work achieves the introduction of security implementation for the P2P and Peer networks proposed with the SATINE project.

The functionalities of the super peers that constitute the SATINE P2P Network are extended to act as the peers of PeerGroup-3, introduced within this work, for the purpose of providing the XML Key Management and Security Token Management services. Thus, the super peers introduced with the SATINE project have also been dedicated to provide additional services of XKMS and WS-Trust.

Similarly, the functionalities of the peers that constitute the SATINE Peer Network are extended to act as the peers of PeerGroup-1 and PeerGroup-2, introduced within this work, for the purpose of providing and invoking Web services in a secure manner. This integration is achieved through the WS-Security programming interface provided by this thesis work, including the generation of asymmetric key pairs (public and private keys), the processing of security assertions stored in WS-SecurityPolicy compliant files, and adding the encryption and digital signature information on the specified element nodes of the constructed messages.

The security mechanism is integrated to the SATINE network as follows:

- Each peer of the SATINE Peer Network creates its own public and private key pairs at start-up by the provided programming interface, and sends the created public key to the trusted super peer of the SATINE P2P Network, which is extended to provide XKMS service, conforming to XKMS-registerPublicKeyRequest message. The sample structure of the generated public-private key pairs through the provided programming interface are as follows:

Public Key:

```
<?xml version="1.0" encoding="UTF-8"?>
<d:KeyInfo xmlns:d="http://www.w3.org/2000/09/xmldsig#">
    <d:KeyName>Public Key</d:KeyName>
    <d:KeyValue xmlns:d="http://www.w3.org/2000/09/xmldsig#">
            <d:RSAKeyValue>

    <d:Modulus>6L0B+NFIEDo5hwzQ+cggEZsFOlYYEtgj5jA8iItUc8AMI0kwqII8I3nv
AlEcOOcvUxBWPfcgghK1HDBYG15KX8zNaK84MMF3i9gqZSP1lewUKSO6bmnRaN
u2TBmpspvDGEalvBWEgnBFZgfIsTJhIOhMCbWRZyqAPPlrsqnP2Mc=</d:Modulus>
                    <d:Exponent
xmlns:d="http://www.w3.org/2000/09/xmldsig#">AQAB</d:Exponent>
            </d:RSAKeyValue>
    </d:KeyValue>
</d:KeyInfo>
```

Private Key:

```
<?xml version="1.0" encoding="UTF-8"?>
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:KeyValue>
            <xkms:RSAKeyPair        xmlns:xkms="http://www.xkms.org/schema/xkms-
2001-01-20">

    <xkms:Modulus>AOi9AfjRSBA6OYcM0PnIIBGbBTpWGBLYI+YwPIiLVHPADC
NJMKiCPCN57wJRHDjnL1MQVj33IIIStRwwWBteSl/MzWivODDBd4vYKmUj9ZXsF
Ckjum5p0WjbtkwZqbKbwxhGpbwVhIJwRWYHyLEyYSDoTAm1kWcqgDz5a7Kpz9jH
</xkms:Modulus>
                    <xkms:PublicExponent>AQAB</xkms:PublicExponent>

    <xkms:PrivateExponent>ALbjsCUC8Iov9vz1SVK/vNOY6ibJeOl2B9/fj/IM6zR6eg9
QeHgyv2dxbix36KGeqkWv1NWk2VrdsIySQOpg0jeuapODYeXAYWLU2u2mxK+C32f
ovbyVrA17r6MYbr9qQCm1h+V8Tfa2O04X/Kj6MOELqpWBUM9NRdGuKEi2Fvph</x
kms:PrivateExponent>

    <xkms:P>APVTE0tjJc2jZh5/YVi9z0dVZiECTok777bC0j/UXtZj44gT+QNNu7xGXr
oNccGE3k31o+PQ9MlXEopW8DA0Tis=</xkms:P>

    <xkms:Q>APLduLfaU376Sa7xK9SruaRVPwDcpNdVux57vl58oLH80pxmzHSFTQ4
O9B6dHplalXRm0NAcG0yirdL8Y1xH7dU=</xkms:Q>

    <xkms:DP>AKR1QrQBDXCjn2vGfN3esLvjVgm+4CNDmNluFUBRABq87+VjbkV
2sOnwSsRzCtVuWxDsISgyBkeLJZSz32SRS+0=</xkms:DP>

    <xkms:DQ>ANi5uWP/pXDzgxtlRfrTf6dBFyb6vvMWIxQR0xDYYEJU3dEJ/zuf1OJ
Nv9Ut2qd46VPliEOeQVJC/aEA7t3jpj0=</xkms:DQ>
```

```
        <xkms:QINV>AzcyEebjvHjDUhxoRo+cD7kX0aOkIVEmU6QNfQZNq6ExIkXrEf4
    6vFDfD/Dfv9ZjvY1JkJCu8GUSsRfx6xGIww==</xkms:QINV>
                </xkms:RSAKeyPair>
        </ds:KeyValue>
    </ds:KeyInfo>
```

- The Web service invoking peer of the SATINE Peer Network requests the secret key information from the trusted super peer of the SATINE P2P Network, which is extended to provide XKMS service, for the session with the Web service providing peer of the SATINE Peer Network. The sample 24-bytes secret key generated through the provided programming interface is:

  Rz□Õ,lpÖkµÈsÜµñ¿‰ ?ÂßŠm^

- The trusted super peers of the SATINE P2P Network store the public key information of the peers of the SATINE Peer Network, and generate and store the secret key information by the provided programming interface for the Web service invoking sessions of the peers of the SATINE Peer Network.
- The Web service invoking peer of the SATINE Peer Network inserts the encryption data based on the secret key information and the digital signature to the constructed Web service message by the provided programming interface. Encryption and signing operations are performed according to the security policies of the Web service providing peer of the SATINE Peer Network, and the provided programming interface is used for this operation. The sample security policy file that is processed to perform encryption and digital signature can be denoted as:

```
<wsp:Policy xmlns:wsp="..." xmlns:wsse="...">
      <wsse:Integrity wsp:Usage="wsp:Required">
            <wsse:Algorithm Type="wsse:AlgSignature" />
            <MessageParts>/InvokeMessage</MessageParts>
      </wsse:Integrity>
      <wsse:Confidentiality wsp:Usage="wsp:Required">
            <wsse:Algorithm Type="wsse:AlgEncryption" />
            <!--
MessageParts>/InvokeMessage/Instance/rdf:RDF/a:Airport</Message
Parts -->

      <MessageParts>/InvokeMessage/Instance</MessageParts>
```

```
   </wsse:Confidentiality>
</wsp:Policy>
```

According to the policy assertions above, the original invoke message is to be encrypted on the "`/InvokeMessage/Instance/rdf:RDF/a:Airport`" element, and the digital signature applied on the element "`/InvokeMessage`" is to be added to the resultant message after encryption. As the following sample invoke message denotes, the messages exchanged are in the form of RDF (Resource Description Framework) data:

```
<?xml version="1.0" encoding="UTF-8"?>
<InvokeMessage>
   <Instance>
         <rdf:RDF xml:base="file:/C:/codes/satine/docs/map/OTA1.rdf"

   xmlns:a="http://www.srdc.metu.edu.tr/~yildiray/ontology/OTADomainOntology_v3.r
dfs#"
               xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
               <a:BookingRequest rdf:ID="a1"
                     a:bookingFlightNo="4437"
                     a:time="050904">
                     <a:hasCompany rdf:resource="#a2"/>
                     <a:bookingProperties rdf:resource="#a4"/>
                     <a:bookingTo rdf:resource="#a9"/>
                     <a:bookingFrom rdf:resource="#a11"/>
                     <a:hasPassenger rdf:resource="#a13"/>
               </a:BookingRequest>
               <a:Company rdf:ID="a2"
                     a:name="TK">
                     <a:contact rdf:resource="#a3"/>
               </a:Company>
               <a:Contact rdf:ID="a3"
                     a:address="METU"
                     a:email="yildiray@srdc.metu.edu.tr"
                     a:phone="2102076"
                     a:fax="2101004">
               </a:Contact>
               <a:Booking rdf:ID="a4"
                     a:bookingClass="Y"
                     a:designCode="">
                     <a:inCabin rdf:resource="#a5"/>
               </a:Booking>
               <a:Cabin rdf:ID="a5"
                     a:cabinType="">
                     <a:hasSeat rdf:resource="#a6"/>
               </a:Cabin>
               <a:Seat rdf:ID="a6">
                     <a:assignedTo rdf:resource="#a7"/>
               </a:Seat>
               <a:Person rdf:ID="a7"
                     a:personName="YILDIRAY">
                     <a:contactInfo rdf:resource="#a8"/>
```

```
                    </a:Person>
                    <a:Contact rdf:ID="a8"
                            a:address=""
                            a:email=""
                            a:phone=""
                            a:fax="">
                    </a:Contact>
                    <a:City rdf:ID="a9"
                            a:cityname="IST">
                            <a:hasAirport rdf:resource="#a10"/>
                    </a:City>
                    <a:Airport rdf:ID="a10"
                            a:airportCode="IST">
                    </a:Airport>
                    <a:City rdf:ID="a11"
                            a:cityname="PAR">
                            <a:hasAirport rdf:resource="#a12"/>
                    </a:City>
                    <a:Airport rdf:ID="a12"
                            a:airportCode="ORY">
                    </a:Airport>
                    <a:Person rdf:ID="a13"
                            a:personName="KABAK">
                            <a:contactInfo rdf:resource="#a14"/>
                    </a:Person>
                    <a:Contact rdf:ID="a14"
                            a:address=""
                            a:email=""
                            a:phone=""
                            a:fax="">
                    </a:Contact>
            </rdf:RDF>
    </Instance>
    <OwlsURI>
    </OwlsURI>
    <WsdlURI>http://www.srdc.metu.edu.tr/~yildiray/amadeus3.wsdl</WsdlURI>
</InvokeMessage>
```

The resultant secure message achieved after applying the operations of encryption and digital signature on the elements defined in the security policy file is as follows:

```
<InvokeMessage>
    <Instance>
            <rdf:RDF                    xml:base="file:/C:/codes/satine/docs/map/OTA1.rdf"
xmlns:a="http://www.srdc.metu.edu.tr/~yildiray/ontology/OTADomainOntology_v3.rdfs#
" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
                    <a:BookingRequest   a:bookingFlightNo="4437"   a:time="050904"
rdf:ID="a1">
                            <a:hasCompany rdf:resource="#a2"/>
                            <a:bookingProperties rdf:resource="#a4"/>
                            <a:bookingTo rdf:resource="#a9"/>
                            <a:bookingFrom rdf:resource="#a11"/>
                    <a:hasPassenger rdf:resource="#a13"/>
               </a:BookingRequest>
               <a:Company a:name="TK" rdf:ID="a2">
                    <a:contact rdf:resource="#a3"/>
               </a:Company>
               <xenc:EncryptedData
Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/><xenc:CipherData><xenc:CipherValue>AmHYIwsuAPAw4PCao2qSkn1Dg95zSsh
35WcBnpUr3uaXb2w0VrD4eFcWc0M1ztB4BECfWal6

   1PUdjRpnu3eG68j+twDMG442iRJf0o/gNLCm1iiKXIFA+I+cvV8AzS0rTP3dLHum/
D9cg16l

   L7ANRrXrd7s9gMV8+C9jEO7DLy1F1gL7mSHJ2FbUInsoPn0yXdU0B/2x+HKT37
vxENHCl93t

   CJnZFsLy0lK9yyJ7rBbC7azJX5zK/irQJEEoCjVM47vE478Df0uhdFr+HmmQ4GpA
SzuagKER

   FWDp0Ao2Sg+0NKO0rby7ca+K2EMWWJW5GFJQVVleEkJ3i+Guc1BXh4lFuIJK
o/bol8vbpJct
                                   vH8=</xenc:CipherValue>
                    </xenc:CipherData>
               </xenc:EncryptedData>
               <a:Booking a:bookingClass="Y" a:designCode="" rdf:ID="a4">
                    <a:inCabin rdf:resource="#a5"/>
               </a:Booking>
               <a:Cabin a:cabinType="" rdf:ID="a5">
                    <a:hasSeat rdf:resource="#a6"/>
               </a:Cabin>
               <a:Seat rdf:ID="a6">
                    <a:assignedTo rdf:resource="#a7"/>
               </a:Seat>
               <xenc:EncryptedData
Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/><xenc:CipherData><xenc:CipherValue>BPm8kR5gKyHkFPylh5g6l6dab3qQwM+
5/GZbQ164R55EulovtQ9Ytkeu8UKwR0UqHy0ojGmf

   pvrQwVkpFrucfMtdc1vP+I9eTC6TR55og6RbmbJ/uaNq+6VMt5zJhNCMtMT7hJ4O
T6VAgHAS

   OSEXzUES80pZuKNuONaH8RsM+29MHk9gYIAU0bIVzKaOaQBFShgQ5PnZL65
sIQxyiiITUKwx

   zs5UolQCvHfocWoCsx14ew73h/sx8SsRLkAEbIZbPi5YZ/MNT7SbWd6pWBmSOZ
5EYsIdI6XO

   V00N3fxUmiYOJKrZuXoR/J9bcgdLWcx+hg4boPuPKHA=</xenc:CipherValue>
                    </xenc:CipherData>
               </xenc:EncryptedData>
```

```
                        <xenc:EncryptedData
Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/><xenc:CipherData><xenc:CipherValue>wFHE/G4oQT1x9nxgSZsFBL2bJX6ssbw
MZPo6CLxos5+WfWuOgdizPPFvENyd70cVbV+3jCPO

    OLkVj2wWOGcClbUhV9umbviW8GL14XoaXLAg48QD3/uwz0sE6kiOu3GzqTHy
QRmZ/d6pYjP3

    2fqPXmx58Xi9SKXCiFnKPA7cYsinZ+o9n20SPk8uDgmAkslxHhOq/5tDU5ZsuFeR
FLpa1THf

    +fNXwjD5yxu+8wK/NktIFj/qV1uzcZgqyPYAxuJU7yBoNxGdGUif7Yi+/Efo3DhTtc
0baoAU

    6e6VDnb3aSqC4u3hxxaD0Q==</xenc:CipherValue>
                                </xenc:CipherData>
                        </xenc:EncryptedData>
                        <a:City a:cityname="IST" rdf:ID="a9">
                                <a:hasAirport rdf:resource="#a10"/>
                        </a:City>
                        <a:Airport a:airportCode="IST" rdf:ID="a10">
                        </a:Airport>
                        <a:City a:cityname="PAR" rdf:ID="a11">
                                <a:hasAirport rdf:resource="#a12"/>
                        </a:City>
                        <a:Airport a:airportCode="ORY" rdf:ID="a12">
                        </a:Airport>
                        <xenc:EncryptedData
Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/><xenc:CipherData><xenc:CipherValue>XXLatS7QOEZWnE/nbdt36MVpw45b/Hb
Oik7aKt2uM86C3RQjeNEJPD0Oabd5CKYHwTBch5lE

    j+zFtAZCd+HJPY3AxpVQ7Crm/lNjGzZrwIhggUBzwEvpTMsTsgEaCj9gmEPcy7Jo
ygWd999Z

    e3xK22SMT9moifvfJw0ZEDWEIufBzbfJKgqb1+IqzbyXnUKYfcypIV2NDvhRMTq
ypQO2XV1k

    6tkzZpyVVBE4kN9VfAZ5W0KNEklg/nHADiYFZ66ltE5poK7EOJoBoW68p3/6fiH
XvbnCynbU

    hLa3aYjmWcud1CRwvgjRz9uLJmMbF32XmLv9SpeiGZA=</xenc:CipherValue>
                                </xenc:CipherData>
                        </xenc:EncryptedData>
                        <xenc:EncryptedData
Type="http://www.w3.org/2001/04/xmlenc#Element"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"><xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/><xenc:CipherData><xenc:CipherValue>lnQSWIZuEd0isesE7waiTEeRcBpFDNIlo
2THHh2SnVVamEaPc78nqFFnZgn5ox74uxJ5jnYO
```

```
      1YC2nbJ0jV3DVqVkqvn27UMUx1y2F5Q6Lc7ldY0B83H9R3m7k4XKK+UVt0w0z
okd5olrAQQ+

      g4xJFOLmJDxhWaJHyLHRPi6/ufV0LXXfNl0Rwzbf9ppBW3Q7cPEXSE0fMm9BN
CKy0JSFNFtb

      imN4ibzef038NYFSmtCPilwIhwLt98UOdbYv/wF5RESFPHLhOKvDSIBimDJEDj
MtKhpkvRyf

      tOBpiR+fi2I6M0Q/NV4Hvw==</xenc:CipherValue>
                                    </xenc:CipherData>
                        </xenc:EncryptedData>
            </rdf:RDF></Instance><OwlsURI>
      </OwlsURI>
      <WsdlURI>http://www.srdc.metu.edu.tr/~yildiray/amadeus3.wsdl</WsdlURI>
      <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"/>
                        <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                        <ds:Reference URI="">
                                    <ds:Transforms>
                                                <ds:Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                                                <ds:Transform
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                                    </ds:Transforms>
                                    <ds:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

      <ds:DigestValue>Hk2vomCsc+G7h8kBFZdUx/qTuHU=</ds:DigestValue>
                        </ds:Reference>
            </ds:SignedInfo>

      <ds:SignatureValue>PKxFIVRKw0GM7rblaVuUmgEd0uO5n1HF9laSuEHWcEy64
MsdhYu45qQ2QQZNh4QmCKf5AaUw

      4UiqbU16umex+KZGn2rkicc/rHMuAh46UlEIdpHarmJ+neYwmC65EncEXbA65/+b
Cuh1ni/J
                        pzxpJP3OTK1mwFNTPJr6DVwFGJ8=</ds:SignatureValue>
      </ds:Signature>
</InvokeMessage>
```

- Having received the secure Web service message, the Web service providing peer of the SATINE Peer Network requests the secret key information from the trusted super peer of the SATINE P2P Network, which is extended to provide XKMS service, for the session with the Web service invoking peer of the SATINE Peer Network, and requests the public key information of the same peer conforming to XKMS-locatePublicKeyRequest message.

- The Web service providing peer of the SATINE Peer Network verifies the digital signature based on the public key information, and decrypts the encrypted data based on the security key information by the provided programming interface.
- Having executed the Web service, the Web service providing peer of the SATINE Peer Network constructs the Web service response message, and inserts the encryption data based on the same secret key information as the one requested and inserts the digital signature based on its own private key information by the provided programming interface.
- The Web service invoking peer of the SATINE Peer Network verifies the digital signature based on the public key information, and decrypts the encrypted data based on the security key information so that this peer can reach the Web service execution result in a secure manner.

CHAPTER VI

RELATED WORK

Web services security is an important research and development area, and several building blocks and standards have been proposed for this topic.

[23] proposes such a mechanism that is achieved over the Internet. This mechanism, introduced by VeriSign, performs XML Key Management Specification interface as a Web service built on the protocol http (hyper-text transfer protocol) and provides a framework for the implementation of WS-Security.

JXTA framework has introduced its own security mechanism [22], which provides whole message integrity and confidentiality only among the communicating peer pairs, and exploits the technology Transport Layer Security (TLS). Hence, the security mechanism introduced by the JXTA framework does not provide a solution required for the exploitation of Web services. For the Web services technology, a more complex mechanism is required in order to involve all the peers (Web service providing, invoking and intermediary peers) within the peer-to-peer network by conforming to the well-accepted specifications introduced for Web services security.

[14] presents the description of a proposed strategy for addressing security within a Web service environment. The proposed architecture is benefitted in this thesis work in order to bring together the security token management and XML Key Management services by delegating these responsibilities to a group of peers acting as a trusted third party for the other peers.

[8], [9], and [10] describe the structures that the XML security technology relies on. These descriptions are exploited within the thesis work to achieve the implementation of the framework that provides the programming interface of WS-Security to be used by the peers.

CHAPTER VII

CONCLUSION AND FUTURE WORK

In order to be able to exploit the Web service technology to its full potential, the specifications related with Web services security should be used in Web service implementations.

In this thesis, realizing the increasing popularity of peer-to-peer computing and the efforts of Project JXTA to provide solutions for peer-to-peer networks, a peer-to-peer approach for providing secure Web service implementation is proposed. For this purpose, the specifications WS-Security, WS-Trust, WS-SecurityPolicy, and XML Key Management Specification are exploited, and brought together with the JXTA framework in order to achieve the introduced system.

The thesis work is realized as a part of the SATINE project funded by the European Commission. The objective of the project is to develop a secure semantic-based interoperability framework for exploiting Web service platforms in conjunction with peer-to-peer networks for the travel industry. This thesis work contributes to the project by enabling the deployment and invocation of Web services in a secure manner in peer-to-peer networks, which is one of the aims of the SATINE project.

There exists a number of issues left as future work, which are:

- The mechanism for distributing the security policy files of Web service providing peers among the Web service invoking peers, which is also currently missing in the specification WS-SecurityPolicy.
- Providing the mechanism and implementation of reliable SOAP messaging among the Web service providing and Web service invoking peers, which is defined by the WS-Reliability specification.
- Extending the introduced system to conform to the other specifications built on the specification WS-Security, which are WS-Privacy defining the preferences and practices of Web services, WS-SecureConversation enabling the SOAP messaging to act like a connection-based approach so that repeating the authentication and policy negotiation is not required for every SOAP message, WS-Authorization defining how Web services manage authorization policies, and WS-Federation enabling associations between security domains.

REFERENCES

[1] World Wide Web Consortium, http://www.w3.org

[2] Web Service Description Language (WSDL), March 2001, http://www.w3.org/TR/wsdl

[3] Simple Object Access Protocol (SOAP), June 2003, http://www.w3.org/TR/SOAP

[4] Universal Description, Discovery, and Integration (UDDI), http://www.uddi.org

[5] Project JXTA, http://www.jxta.org

[6] Milojicic D., Kalogeraki V., Lukose R., Nagaraja K., Pruyne J., Richard B., Rollins S., Xu Z., “Peer-to-peer Computing”, HP Laboratories Palo Alto, March 2002

[7] Li Gong, “JXTA: A Network Programming Environment”, Sun Microsystems, June 2001

[8] Simon E., Madsen P., Adams C., “An Introduction To XML Digital Signatures”, August 2001

[9] Mactaggart M., “Enabling XML Security”, IBM DeveloperWorks, September 2001

[10] Salz R., “Understanding XML Digital Signature”, Microsoft Corporation, July 2003

[11] Seely S., “Understanding WS-Security”, Microsoft Corporation, October 2002

[12] Atkinson B., Della-Libera G., Hada S., Hondo M., Hallam-Baker P., Kaler C., Klein J., LaMacchia B., Leach P., Manferdelli J., Maruyama H., Nadalin A., Nagaratnam N., Prafullchandra H., Shewchuk J., Simon

D., "Web Services Security Version 1.0", IBM & Microsoft & VeriSign, April 2002

[13] Geer D., "Taking Steps To Secure Web Services", IEEE Computer, October 2003

[14] Joint white paper, "Security In a Web Services World: A Proposed Architecture and Roadmap", IBM Corporation & Microsoft Corporation, April 2002

[15] Anderson S., Bohren J., Boubez T., Chanliau M., Della-Libera G., Dixon B., Garg P., Gravengaard E., Gudgin M., Hallam-Baker P., Hondo M., Kaler C., Lockhart H., Martherus R., Maruyama H., Mishra P., Nadalin A., Nagaratnam N., Nash A., Philpott R., Platt D., Prafullchandra H., Sahu M., Shewchuk J., Simon D., Srinivas D., Waingold E., Waite D., Zolfonoon R., "Web Services Trust Language Version 1.1", BEA Systems, Computer Associates International, IBM, Layer 7 Technologies, Microsoft, Netegrity, Oblix, OpenNetwork Technologies, Ping Identity, Reactivity, RSA Security, VeriSign, Westbridge Technology, May 2004

[16] Box D., Curbera F., Hondo M., Kaler C., Langworthy D., Nadalin A., Nagaratnam N., Nottingham M., Riegen C., Shewchuk J., "Web Services Policy Framework", BEA Systems, IBM, Microsoft, SAP, May 2003

[17] Della-Libera G., Hallam-Baker P., Hondo M., Janzcuk T., Kaler C., Maruyama H., Nadalin A., Nagaratnam N., Nash A., Philpott R., Prafullchandra H., Shewchuk J., Waingold E., Zolfonoon R., "Web Services Security Policy Language Version 1.0", IBM, Microsoft, RSA Security, VeriSign, December 2002

[18] Skonnard A., "Understanding WS-Policy", Skonnard Consulting, August 2003

[19] Nagappan R., Skoczylas R., Sriganesh R., "Developing Java Web Services", John Wiley & Sons, 2003

[20] Dournaee B., "XML Security", McGraw-Hill/Osborne, 2002

[21] Gradecki J., "Mastering JXTA: Building Java Peer-to-Peer Applications", John Wiley & Sons, 2002

[22] "Security and Project JXTA", Sun Microsystems, January 2002

[23] The VeriSign Company, http://www.verisign.com

[24] The SATINE Project, http://www.srdc.metu.edu.tr/webpage/projects/satine/

[25] Dogac A., Kabak Y., Laleci G., Sinir S., Yildiz A., Kirbas S., Gurcan Y., "Semantically Enriched Web Services for the Travel Industry", ACM Sigmod Record, Vol. 33, No. 3, September 2004

[26] Dogac A., Kabak Y., Laleci G., Sinir S., Yildiz A., Tumer A., "SATINE Project: Exploiting Web Services in the Travel Industry", eChallenges 2004 (e-2004), October 2004

[27] SATINE Consortium, Demo, "SATINE Project: A Semantically-enriched Web Service Platform for the Travel Industry", eChallenges 2004 (e-2004), October 2004

[28] Flugge M., Tourtchaninova D., "Ontology-derived Activity Components for Composing Travel Web Services", The International Workshop on Semantic Web Technologies in Electronic Business (SWEB2004), October 2004

APPENDIX A

DESCRIPTION FILES OF THE WEB SERVICES

The WSDL description file of the XML Key Management Web service, which is provided by the peers of PeerGroup-3, is presented as follows:

```
<?xml version="1.0"?>

<definitions name="XMLKeyManagement"
xmlns="http://schemas.xmlsoap.org/wsdl/"
 xmlns:s="http://www.w3.org/2000/10/XMLSchema"
 xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xkms="http://www.xkms.org/schema/xkms-2001-01-20"
 xmlns:tns="http://www.xkms.org/schema/xkms-2001-01-20"
 targetNamespace="http://www.xkms.org/schema/xkms-2001-01-20">

  <types>

<schema xmlns="http://www.w3.org/2001/XMLSchema"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
       targetNamespace="http://www.w3.org/2000/09/xmldsig#"
       version="0.1" elementFormDefault="qualified">

<!-- Basic Types Defined for Signatures -->

<simpleType name="CryptoBinary">
  <restriction base="base64Binary">
  </restriction>
</simpleType>

<!-- Start Signature -->
```

```
<element name="Signature" type="ds:SignatureType"/>
<complexType name="SignatureType">
  <sequence>
    <element ref="ds:SignedInfo"/>
    <element ref="ds:SignatureValue"/>
    <element ref="ds:KeyInfo" minOccurs="0"/>
    <element ref="ds:Object" minOccurs="0"
maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>

  <element name="SignatureValue" type="ds:SignatureValueType"/>
  <complexType name="SignatureValueType">
    <simpleContent>
      <extension base="base64Binary">
        <attribute name="Id" type="ID" use="optional"/>
      </extension>
    </simpleContent>
  </complexType>

<!-- Start SignedInfo -->

<element name="SignedInfo" type="ds:SignedInfoType"/>
<complexType name="SignedInfoType">
  <sequence>
    <element ref="ds:CanonicalizationMethod"/>
    <element ref="ds:SignatureMethod"/>
    <element ref="ds:Reference" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>

  <element name="CanonicalizationMethod"
type="ds:CanonicalizationMethodType"/>
  <complexType name="CanonicalizationMethodType" mixed="true">
    <sequence>
      <any namespace="##any" minOccurs="0"
maxOccurs="unbounded"/>
      <!-- (0,unbounded) elements from (1,1) namespace -->
    </sequence>
    <attribute name="Algorithm" type="anyURI" use="required"/>
  </complexType>

  <element name="SignatureMethod"
type="ds:SignatureMethodType"/>
  <complexType name="SignatureMethodType" mixed="true">
    <sequence>
      <element name="HMACOutputLength" minOccurs="0"
type="ds:HMACOutputLengthType"/>
      <any namespace="##other" minOccurs="0"
maxOccurs="unbounded"/>
      <!-- (0,unbounded) elements from (1,1) external namespace
-->
    </sequence>
    <attribute name="Algorithm" type="anyURI" use="required"/>
  </complexType>

<!-- Start Reference -->

<element name="Reference" type="ds:ReferenceType"/>
<complexType name="ReferenceType">
  <sequence>
    <element ref="ds:Transforms" minOccurs="0"/>
    <element ref="ds:DigestMethod"/>
    <element ref="ds:DigestValue"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
  <attribute name="URI" type="anyURI" use="optional"/>
  <attribute name="Type" type="anyURI" use="optional"/>
</complexType>

  <element name="Transforms" type="ds:TransformsType"/>
  <complexType name="TransformsType">
    <sequence>
      <element ref="ds:Transform" maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <element name="Transform" type="ds:TransformType"/>
  <complexType name="TransformType" mixed="true">
    <choice minOccurs="0" maxOccurs="unbounded">
      <any namespace="##other" processContents="lax"/>
      <!-- (1,1) elements from (0,unbounded) namespaces -->
      <element name="XPath" type="string"/>
    </choice>
    <attribute name="Algorithm" type="anyURI" use="required"/>
  </complexType>

<!-- End Reference -->

<element name="DigestMethod" type="ds:DigestMethodType"/>
<complexType name="DigestMethodType" mixed="true">
  <sequence>
    <any namespace="##other" processContents="lax"
minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>

<element name="DigestValue" type="ds:DigestValueType"/>
<simpleType name="DigestValueType">
  <restriction base="base64Binary"/>
</simpleType>

<!-- End SignedInfo -->

<!-- Start KeyInfo -->

<element name="KeyInfo" type="ds:KeyInfoType"/>

<complexType name="KeyInfoType" mixed="true">
```

```
  <sequence>
<!--    <element ref="ds:KeyName"/>     -->
    <element ref="ds:KeyName"/>
    <element ref="ds:KeyValue"/>
    <element ref="ds:RetrievalMethod"/>
    <element ref="ds:X509Data"/>
    <element ref="ds:PGPData"/>
    <element ref="ds:SPKIData"/>
    <element ref="ds:MgmtData"/>
    <any processContents="lax" namespace="##other"/>
    <!-- (1,1) elements from (0,unbounded) namespaces -->
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>

  <element name="KeyName" type="string"/>
  <element name="MgmtData" type="string"/>

  <element name="KeyValue" type="ds:KeyValueType"/>
  <complexType name="KeyValueType" mixed="true">
   <choice>
     <element ref="ds:DSAKeyValue"/>
     <element ref="ds:RSAKeyValue"/>
     <any namespace="##other" processContents="lax"/>
   </choice>
  </complexType>

  <element name="RetrievalMethod"
type="ds:RetrievalMethodType"/>
  <complexType name="RetrievalMethodType">
    <sequence>
      <element name="Transforms" type="ds:TransformsType"
minOccurs="0"/>
    </sequence>
    <attribute name="URI" type="anyURI"/>
    <attribute name="Type" type="anyURI" use="optional"/>
  </complexType>

<!-- Start X509Data -->

<element name="X509Data" type="ds:X509DataType"/>
<complexType name="X509DataType">
  <sequence maxOccurs="unbounded">
    <choice>
      <element name="X509IssuerSerial"
type="ds:X509IssuerSerialType"/>
      <element name="X509SKI" type="base64Binary"/>
      <element name="X509SubjectName" type="string"/>
      <element name="X509Certificate" type="base64Binary"/>
      <element name="X509CRL" type="base64Binary"/>
      <any namespace="##other" processContents="lax"/>
    </choice>
  </sequence>
</complexType>

<complexType name="X509IssuerSerialType">
```

```
  <sequence>
    <element name="X509IssuerName" type="string"/>
    <element name="X509SerialNumber" type="integer"/>
  </sequence>
</complexType>

<!-- End X509Data -->

<!-- Begin PGPData -->

<element name="PGPData" type="ds:PGPDataType"/>
<complexType name="PGPDataType">
  <choice>
    <sequence>
      <element name="PGPKeyID" type="base64Binary"/>
      <element name="PGPKeyPacket" type="base64Binary"
minOccurs="0"/>
      <any namespace="##other" processContents="lax"
minOccurs="0"
       maxOccurs="unbounded"/>
    </sequence>
  </choice>
</complexType>

<!-- End PGPData -->

<!-- Begin SPKIData -->

<element name="SPKIData" type="ds:SPKIDataType"/>
<complexType name="SPKIDataType">
  <sequence maxOccurs="unbounded">
    <element name="SPKISexp" type="base64Binary"/>
    <any namespace="##other" processContents="lax"
minOccurs="0"/>
  </sequence>
</complexType>

<!-- End SPKIData -->

<!-- End KeyInfo -->

<!-- Start Object (Manifest, SignatureProperty) -->

<element name="Object" type="ds:ObjectType"/>
<complexType name="ObjectType" mixed="true">
  <sequence minOccurs="0" maxOccurs="unbounded">
    <any namespace="##any" processContents="lax"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
  <attribute name="MimeType" type="string" use="optional"/> <!-
- add a grep facet -->
  <attribute name="Encoding" type="anyURI" use="optional"/>
</complexType>

<element name="Manifest" type="ds:ManifestType"/>
<complexType name="ManifestType">
```

```
  <sequence>
    <element ref="ds:Reference" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>

<element name="SignatureProperties"
type="ds:SignaturePropertiesType"/>
<complexType name="SignaturePropertiesType">
  <sequence>
    <element ref="ds:SignatureProperty" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>

   <element name="SignatureProperty"
type="ds:SignaturePropertyType"/>
   <complexType name="SignaturePropertyType" mixed="true">
     <choice maxOccurs="unbounded">
       <any namespace="##other" processContents="lax"/>
       <!-- (1,1) elements from (1,unbounded) namespaces -->
     </choice>
     <attribute name="Target" type="anyURI" use="required"/>
     <attribute name="Id" type="ID" use="optional"/>
   </complexType>

<!-- End Object (Manifest, SignatureProperty) -->

<!-- Start Algorithm Parameters -->

<simpleType name="HMACOutputLengthType">
  <restriction base="integer"/>
</simpleType>

<!-- Start KeyValue Element-types -->

<element name="DSAKeyValue" type="ds:DSAKeyValueType"/>
<complexType name="DSAKeyValueType">
  <sequence>
    <sequence minOccurs="0">
      <element name="P" type="ds:CryptoBinary"/>
      <element name="Q" type="ds:CryptoBinary"/>
    </sequence>
    <element name="J" type="ds:CryptoBinary" minOccurs="0"/>
    <element name="G" type="ds:CryptoBinary" minOccurs="0"/>
    <element name="Y" type="ds:CryptoBinary"/>
    <sequence minOccurs="0">
      <element name="Seed" type="ds:CryptoBinary"/>
      <element name="PgenCounter" type="ds:CryptoBinary"/>
    </sequence>
  </sequence>
</complexType>


<element name="RSAKeyValue" type="ds:RSAKeyValueType"/>
<complexType name="RSAKeyValueType">
```

```
  <sequence>
    <element name="Modulus" type="ds:CryptoBinary"/>
    <element name="Exponent" type="ds:CryptoBinary"/>
  </sequence>
</complexType>

<!-- End KeyValue Element-types -->

<!-- End Signature -->

</schema>

<schema attributeFormDefault="qualified"
elementFormDefault="qualified"
 targetNamespace="http://www.xkms.org/schema/xkms-2001-01-20"
 xmlns="http://www.w3.org/2001/XMLSchema">
 <import namespace="http://www.w3.org/2000/09/xmldsig#"/>
   <element name="Recover" type="xkms:RecoverType"/>
   <element name="Revoke" type="xkms:RevokeType"/>
   <element name="Locate" type="xkms:LocateType"/>
   <element name="Register" type="xkms:RegisterType"/>
   <element name="Validate" type="xkms:ValidateType"/>
   <element name="RegisterResult"
type="xkms:RegisterResultType"/>
   <element name="RecoverResult"
type="xkms:RecoverResultType"/>
   <element name="RevokeResult" type="xkms:RevokeResultType"/>
   <element name="LocateResult" type="xkms:LocateResultType"/>
   <element name="ValidateResult"
type="xkms:ValidateResultType"/>
   <complexType name="LocateType">
      <sequence>
         <element minOccurs="0" name="TransactionID"
type="string"/>
         <element name="Query" type="xkms:KeyInfoType"/>
         <element minOccurs="0" name="Respond"
type="xkms:RespondType"/>
      </sequence>
   </complexType>
   <complexType name="LocateResultType">
      <sequence>
         <element minOccurs="0" name="TransactionID"
type="string"/>
         <element name="Result" type="xkms:ResultCodeType"/>
         <element minOccurs="0" name="Answer"
type="xkms:LocateResultAnswerType"/>
         <element minOccurs="0" name="ErrorInfo"
type="xkms:ErrorInfoType"/>
         <element maxOccurs="1" minOccurs="0"
ref="ds:Signature"/>
      </sequence>
      <attribute name="Id" type="ID" use="optional"/>
   </complexType>
   <complexType name="ValidateType">
      <sequence>
         <element name="Query" type="xkms:KeyBindingType"/>
         <element minOccurs="0" name="Respond"
type="xkms:RespondType"/>
      </sequence>
   </complexType>
   <complexType name="ValidateResultType">
      <sequence>
         <element name="Result" type="xkms:ResultCodeType"/>
         <element minOccurs="0" name="Answer"
type="xkms:ValidateResultAnswerType"/>
         <element minOccurs="0" name="ErrorInfo"
type="xkms:ErrorInfoType"/>
         <element maxOccurs="1" minOccurs="0"
ref="ds:Signature"/>
      </sequence>
      <attribute name="Id" type="ID" use="optional"/>
   </complexType>
   <complexType name="ValidateResultAnswerType">
      <sequence>
         <element name="KeyBinding"
type="xkms:KeyBindingType"/>
      </sequence>
   </complexType>
   <complexType name="RegisterType">
      <sequence>
         <element name="Prototype" type="xkms:KeyBindingType"/>
         <element name="AuthInfo" type="xkms:AuthInfoType"/>
         <element name="Respond" type="xkms:RespondType"/>
      </sequence>
   </complexType>
   <complexType name="RegisterResultType">
      <sequence>
         <element name="Result" type="xkms:ResultCodeType"/>
         <element minOccurs="0" name="Answer"
type="xkms:RegisterResultAnswerType"/>
         <element minOccurs="0" name="Private"
type="xkms:PrivateType"/>
         <element minOccurs="0" name="ErrorInfo"
type="xkms:ErrorInfoType"/>
         <element maxOccurs="1" minOccurs="0"
ref="ds:Signature"/>
      </sequence>
      <attribute name="Id" type="ID" use="optional"/>
   </complexType>
   <complexType name="RegisterResultAnswerType">
      <sequence>
         <element name="KeyBinding"
type="xkms:KeyBindingType"/>
      </sequence>
   </complexType>
   <complexType name="RecoverType">
      <sequence>
         <element ref="xkms:Register"/>
      </sequence>
   </complexType>
   <complexType name="RecoverResultType">
      <sequence>
```

```
         <element ref="xkms:RegisterResult"/>
      </sequence>
   </complexType>
   <complexType name="RevokeType">
      <sequence>
         <element ref="xkms:Register"/>
      </sequence>
   </complexType>

   <complexType name="PrivateType">
      <sequence>
         <any maxOccurs="unbounded" processContents="lax"
            minOccurs="0" namespace="##any"/>
      </sequence>
      <anyAttribute namespace="##any" processContents="lax"/>
   </complexType>

   <complexType name="RevokeResultType">
      <sequence>
         <element ref="xkms:RegisterResult"/>
      </sequence>
   </complexType>
   <complexType name="ErrorInfoType">
      <sequence>
         <element name="ErrorDescription" type="string"/>
         <element minOccurs="0" name="ErrorActor"
type="string"/>
         <element minOccurs="0" name="ErrorDetail"
type="xkms:ErrorDetailType"/>
      </sequence>
      <attribute name="errorCode" type="string"
use="required"/>
   </complexType>
   <complexType name="ErrorDetailType">
      <sequence>
         <any namespace="##other" processContents="strict"/>
      </sequence>
      <anyAttribute namespace="##other"
processContents="strict"/>
   </complexType>
   <simpleType name="ErrorCodeType">
      <restriction base="string">
         <enumeration value="Client"/>
         <enumeration value="Server"/>
         <enumeration value="Client.MalformedKeyNamePassed"/>
         <enumeration value="Client.InvalidPassPhraseAuth"/>
         <enumeration value="*"/>
      </restriction>
   </simpleType>
   <complexType name="KeyBindingType">
      <sequence>
         <element minOccurs="0" name="TransactionID"
type="string"/>
         <element name="Status"
type="xkms:AssertionStatusType"/>
```

```
         <element maxOccurs="unbounded" minOccurs="0"
name="KeyID" type="string"/>
         <element minOccurs="0" ref="ds:KeyInfo"/>
         <element minOccurs="0" name="PassPhrase"
type="string"/>
         <element minOccurs="0" name="ProcessInfo"
type="xkms:ProcessInfoType"/>
         <element minOccurs="0" name="ValidityInterval"
type="xkms:ValidityIntervalType"/>
         <element maxOccurs="unbounded" minOccurs="0"
name="KeyUsage" type="xkms:KeyUsageType"/>
         <element minOccurs="0" name="Private"
type="xkms:PrivateType"/>
      </sequence>
      <attribute name="Id" type="ID" use="optional"/>
   </complexType>
   <complexType name="KeyInfoType">
      <sequence>
         <element ref="ds:KeyInfo"/>
      </sequence>
   </complexType>
   <complexType name="RespondType">
      <sequence>
         <element maxOccurs="unbounded" minOccurs="0"
name="string" type="xkms:RespondEnum"/>
      </sequence>
   </complexType>
   <simpleType name="RespondEnum">
      <restriction base="string">
         <enumeration value="KeyName"/>
         <enumeration value="KeyValue"/>
         <enumeration value="X509Cert"/>
         <enumeration value="X509Chain"/>
         <enumeration value="X509CRL"/>
         <enumeration value="OCSP"/>
         <enumeration value="RetrievalMethod"/>
         <enumeration value="MgmtData"/>
         <enumeration value="PGPData"/>
         <enumeration value="PGPWeb"/>
         <enumeration value="SPKIData"/>
         <enumeration value="Multiple"/>
         <enumeration value="Private"/>
         <enumeration value="ValidityInterval"/>
         <enumeration value="KeyUsage"/>
         <enumeration value="Status"/>
         <enumeration value="SignedResult"/>
      </restriction>
   </simpleType>
   <complexType name="LocateResultAnswerType">
      <sequence>
         <element maxOccurs="unbounded" minOccurs="0"
ref="ds:KeyInfo"/>
      </sequence>
   </complexType>
   <simpleType name="ResultCodeType">
      <restriction base="string">
```

```
         <enumeration value="Success"/>
         <enumeration value="NoMatch"/>
         <enumeration value="NotFound"/>
         <enumeration value="Incomplete"/>
         <enumeration value="Failure"/>
         <enumeration value="Refused"/>
         <enumeration value="Pending"/>
      </restriction>
   </simpleType>
   <simpleType name="AssertionStatusType">
      <restriction base="string">
         <enumeration value="Valid"/>
         <enumeration value="Invalid"/>
         <enumeration value="Indeterminate"/>
      </restriction>
   </simpleType>
   <complexType name="ProcessInfoType">
      <sequence maxOccurs="unbounded" minOccurs="0">
         <any namespace="##other"/>
      </sequence>
   </complexType>
   <complexType name="ValidityIntervalType">
      <sequence>
         <element minOccurs="0" name="NotBefore"
type="timeInstant"/>
         <element minOccurs="0" name="NotAfter"
type="timeInstant"/>
      </sequence>
   </complexType>
   <simpleType name="KeyUsageType">
      <restriction base="string">
         <enumeration value="Encryption"/>
         <enumeration value="Signature"/>
         <enumeration value="Exchange"/>
      </restriction>
   </simpleType>
   <complexType name="AuthInfoType">
      <choice>
         <element name="AuthUserInfo"
type="xkms:AuthUserInfoType"/>
         <element name="AuthServerInfo"
type="xkms:AuthServerInfoType"/>
      </choice>
   </complexType>
   <complexType name="ProofOfPossessionType">
      <sequence>
         <element minOccurs="0" ref="ds:Signature"/>
      </sequence>
   </complexType>
   <complexType name="KeyBindingAuthType">
      <sequence>
         <element minOccurs="0" ref="ds:Signature"/>
      </sequence>
   </complexType>
   <complexType name="AuthUserInfoType">
      <sequence>
```

```
         <element minOccurs="0" name="ProofOfPossession"
type="xkms:ProofOfPossessionType"/>
         <element minOccurs="0" name="KeyBindingAuth"
type="xkms:KeyBindingAuthType"/>
         <element minOccurs="0" name="PassPhraseAuth"
type="string"/>
      </sequence>
   </complexType>
   <complexType name="AuthServerInfoType">
      <sequence>
         <element minOccurs="0" name="KeyBindingAuth"
type="xkms:KeyBindingAuthType"/>
         <element minOccurs="0" name="PassPhraseAuth"
type="string"/>
      </sequence>
   </complexType>
</schema>
  </types>

  <message name="Register">
    <part name="body" element="xkms:Register"/>
  </message>

  <message name="RegisterResult">
    <part name="body" element="xkms:RegisterResult"/>
  </message>

  <message name="Validate">
    <part name="body" element="xkms:Validate"/>
  </message>

  <message name="ValidateResult">
    <part name="body" element="xkms:ValidateResult"/>
  </message>

  <message name="Locate">
    <part name="body" element="xkms:Locate"/>
  </message>

  <message name="LocateResult">
    <part name="body" element="xkms:LocateResult"/>
  </message>

  <portType name="KeyServicePortType">
    <operation name="Register">
      <input message="tns:Register"/>
      <output message="tns:RegisterResult"/>
      <fault message="tns:RegisterResult"/>
    </operation>

    <operation name="Validate">
      <input message="tns:Validate"/>
      <output message="tns:ValidateResult"/>
      <fault message="tns:ValidateResult"/>
    </operation>
    <operation name="Locate">
      <input message="tns:Locate"/>
      <output message="tns:LocateResult"/>
      <fault message="tns:LocateResult"/>
    </operation>
  </portType>

  <binding name="KeyServiceSoapBinding"
type="tns:KeyServicePortType">
    <soap:binding
transport="http://schemas.xmlsoap.org/soap/http"
style="document"/>

    <operation name="Register">
      <soap:operation
soapAction="http://www.xkms.org/schema/xkms-2001-01-
20#Register" style="document"/>
      <input message="tns:Register">
        <soap:body parts="body" use="literal"/>
      </input>
      <output message="tns:RegisterResult">
        <soap:body parts="body" use="literal"/>
      </output>
    </operation>

    <operation name="Validate">
      <soap:operation
soapAction="http://www.xkms.org/schema/xkms-2001-01-
20#Validate" style="document"/>
      <input message="tns:Validate">
        <soap:body parts="body" use="literal"/>
      </input>
      <output message="tns:ValidateResult">
        <soap:body parts="body" use="literal"/>
      </output>
    </operation>

    <operation name="Locate">
      <soap:operation
soapAction="http://www.xkms.org/schema/xkms-2001-01-20#Locate"
style="document"/>
      <input message="tns:Locate">
        <soap:body parts="body" use="literal"/>
      </input>
      <output message="tns:LocateResult">
        <soap:body parts="body" use="literal"/>
      </output>
    </operation>
  </binding>

  <binding name="KeyServiceHttpPostBinding"
type="tns:KeyServicePortType">
    <http:binding verb="POST"/>

    <operation name="Register">
      <http:operation location="xkms/Acceptor.nano"/>
      <input message="tns:Register">
        <mime:content parts="body" type="text/xml"/>
      </input>
      <output message="tns:RegisterResult">
        <mime:content parts="body" type="text/xml"/>
      </output>
    </operation>

    <operation name="Validate">
      <http:operation location="xkms/Acceptor.nano"/>
      <input message="tns:Validate">
        <mime:content parts="body" type="text/xml"/>
      </input>
      <output message="tns:ValidateResult">
        <mime:content parts="body" type="text/xml"/>
      </output>
    </operation>

    <operation name="Locate">
      <http:operation location="xkms/Acceptor.nano"/>
      <input message="tns:Locate">
        <mime:content parts="body" type="text/xml"/>
      </input>
      <output message="tns:LocateResult">
        <mime:content parts="body" type="text/xml"/>
      </output>
    </operation>
  </binding>

  <service name="XMLKeyManagementService">
      <documentation>Verisign's XML Key Management Service
(XKMS)</documentation>

    <port name="KeyServiceSoapPort"
binding="tns:KeyServiceSoapBinding">
      <soap:address
location="http://xkms.verisign.com/xkms/Acceptor.nano"/>
    </port>

    <port name="KeyServiceHttpPostPort"
binding="tns:KeyServiceHttpPostBinding">
      <http:address location="http://xkms.verisign.com/"/>
    </port>
  </service>

  <service name="PilotXMLKeyManagementService">
      <documentation>Verisign's Pilot XML Key Management
Service (XKMS)</documentation>

    <port name="KeyServiceSoapPort"
binding="tns:KeyServiceSoapBinding">
      <soap:address location="http://pilot-
xkms.verisign.com/xkms/Acceptor.nano"/>
    </port>
    <port name="KeyServiceHttpPostPort"
binding="tns:KeyServiceHttpPostBinding">
      <http:address location="http://pilot-
xkms.verisign.com/"/>
    </port>
  </service>
</definitions>
```

The WSDL description file of the Security Token Management Web service, which is provided by the peers of PeerGroup-3, is presented as follows:

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

targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>
   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>

   <binding name="Hello_Binding" type="tns:Hello_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="sayHello">
         <soap:operation soapAction="sayHello"/>
         <input>
            <soap:body

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </input>
         <output>
            <soap:body

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </output>
      </operation>
   </binding>

   <service name="Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
         <soap:address

location="http://localhost:8080/soap/servlet/rpcrouter"/>
      </port>
   </service>
</definitions>
```

The WSDL description file of the Purchase Order Web service, which is provided by the peers of PeerGroup-1 for the demonstrational purpose and invoked by the peers of PeerGroup-2 in a secure manner, is presented as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="OrderService"
   targetNamespace="http://ilhami/wsdl/OrderService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://ilhami/wsdl/OrderService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="OrderRequest">
      <part name="item_name" type="xsd:string"/>
      <part name="price" type="xsd:string"/>
      <part name="card_name" type="xsd:string"/>
      <part name="expiration" type="xsd:string"/>
      <part name="card_number" type="xsd:string"/>
      <part name="card_limit" type="xsd:string"/>
   </message>
   <message name="OrderResponse">
      <part name="status" type="xsd:string"/>
   </message>

   <portType name="Order_PortType">
      <operation name="orderRequest">
         <input message="tns:OrderRequest"/>
         <output message="tns:OrderResponse"/>
      </operation>
   </portType>

   <binding name="Order_Binding" type="tns:Order_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="orderRequest">
         <soap:operation soapAction="orderRequest"/>
         <input>
            <soap:body

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:orderservice"
               use="encoded"/>
         </input>
         <output>
            <soap:body

encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:orderservice"
               use="encoded"/>
         </output>
      </operation>
   </binding>

   <service name="Order_Service">
      <documentation>WSDL File for OrderService</documentation>
      <port binding="tns:Order_Binding" name="Order_Port">
         <soap:address

location="http://localhost:8080/soap/servlet/rpcrouter"/>
      </port>
   </service>
</definitions>
```

APPENDIX B

FORMATS OF SOAP MESSAGES FOR WEB SERVICES INVOCATION

The sample formats of the XML Key Management Web service that denote the public key location request and response are presented in Chapter III. The following SOAP message format represents the revocation request for some public key information:

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

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <k:Register xmlns:k="http://www.xkms.org/schema/xkms-
2001-01-20">
         <k:Prototype Id="refId_1">
            <k:TransactionID>cde285f0-1f14-11d6-b840-
3beb2501bc66</k:TransactionID>
            <k:Status>Invalid</k:Status>
            <d:KeyInfo
xmlns:d="http://www.w3.org/2000/09/xmldsig#">
               <d:KeyValue
xmlns:d="http://www.w3.org/2000/09/xmldsig#">
                  <d:RSAKeyValue>

<d:Modulus>rlc2VJ+acRniqpF8dh5N2qyUYOkne5C257v5Rme13KGl4B7SmAa3
40uBBicdx2RkfyHpW0q3Nm/iY7h8UY1+Cw==</d:Modulus>
                     <d:Exponent
xmlns:d="http://www.w3.org/2000/09/xmldsig#">AQAB</d:Exponent>
                  </d:RSAKeyValue>
               </d:KeyValue>
            </d:KeyInfo>

<k:PassPhrase>SxbPA0qih0NsrYU8ji+gWOhwy7c=</k:PassPhrase>
         </k:Prototype>
         <k:AuthInfo>
            <k:AuthUserInfo/>
         </k:AuthInfo>
         <k:Respond>
            <k:string>KeyName</k:string>
            <k:string>KeyValue</k:string>
         </k:Respond>
      </k:Register>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

As for the response of the revocation request above, the following SOAP message format represents the revocation response for public key information:

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

<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Body>
      <RegisterResult Id="refId_0"
xmlns="http://www.xkms.org/schema/xkms-2001-01-20"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:xkms="http://www.xkms.org/schema/xkms-2001-01-20">
         <Result>Success</Result>
         <Answer>
            <KeyBinding>
               <TransactionID>cde285f0-1f14-11d6-b840-
3beb2501bc66</TransactionID>
               <Status>Invalid</Status>

<KeyID>http://xkms.verisign.com/key?company=VeriSign&amp;depart
ment=XKMS Test&amp;CN=N100055
XKMSTEST&amp;issuer_serial=85dfd14e8c4ecb4adeec353ca4c7b196</Ke
yID>
               <KeyInfo
xmlns="http://www.w3.org/2000/09/xmldsig#">

<KeyName>http://xkms.verisign.com/key?company=VeriSign&amp;depa
rtment=XKMS Test&amp;CN=N100055
XKMSTEST&amp;issuer_serial=85dfd14e8c4ecb4adeec353ca4c7b196</Ke
yName>
                  <RetrievalMethod
Type="http://service.xmltrustcenter.org/XKMS"
URI="http://xkms.verisign.com/xkms/Acceptor.nano"/>
                  <d:KeyValue
xmlns:d="http://www.w3.org/2000/09/xmldsig#"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                     <d:RSAKeyValue>
```

```
<d:Modulus>AK5XNlSfmnEZ4qqRfHYeTdqslGDpJ3uQtue7+UZntdyhpeAe0pgG
t+NLgQYnHcdkZH8h6VtKtzZv4mO4fFGNfgs=</d:Modulus>
                        <d:Exponent>AQAB</d:Exponent>
                     </d:RSAKeyValue>
                  </d:KeyValue>
                  <dsig:X509Data
xmlns:d="http://www.w3.org/2000/09/xmldsig#"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                     <dsig:X509IssuerSerial>
                        <dsig:X509IssuerName>CN=VeriSign Class
2 OnSite Individual TEST CA, OU=For Test Purposes Only,
OU=VeriSign Class 2 OnSite Individual CA,
O=VeriSign</dsig:X509IssuerName>

<dsig:X509SerialNumber>113123383192473107369800977913946522846<
/dsig:X509SerialNumber>
                     </dsig:X509IssuerSerial>
                     <dsig:X509SubjectName>CN=n100055 xkmstest,
OU=&quot;www.verisign.com/repository/CPS Incorp. by
Ref.,LIAB.LTD(c)96&quot;, OU=XKMS Test,
O=VeriSign</dsig:X509SubjectName>

<dsig:X509SKI>MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAK5XNlSfmnEZ4qqRf
HYeTdqslGDpJ3uQtue7+UZntdyhpeAe0pgGt+NLgQYnHcdkZH8h6VtKtzZv4mO4
fFGNfgsCAwEAAQ==</dsig:X509SKI>

<dsig:X509Certificate>MIIDVTCCAv+gAwIBAgIQVRrFZ3hUx8OXAVXjzrtE3
jANBgkqhkiG9w0BAQQFADCBmTERMA8GA1UEChMIVmVyaVNpZ24xLjAsBgNVBAsT
JVZlcmlTaWduIENsYXNzIDIgT25TaXRlIEluZGl2aWR1YWwgQ0ExHzAdBgNVBAs
TFkZvciBUZXN0IFB1cnBvc2VzIE9ubHkxMzAxBgNVBAMTKlZlcmlTaWduIENsYX
NzIDIgT25TaXRlIEluZGl2aWR1YWwgVEVTVCBDQTAeFw0wMjAyMTEwMDAwMDBaF
w0wMzAyMTEyMzU5NTlaMIGKMREwDwYDVQQKFAhWZXJpU2lnbjESMBAGA1UECxQJ
WEtNUyBUZXN0MUYwRAYDVQQLEz13d3cudmVyaXNpZ24uY29tL3JlcG9zaXRvcnk
vQ1BTIEluY29ycC4gYnkgUmVmLixMSUFCLkxURChjKTk2MRkwFwYDVQQDExBuMT
AwMDU1IHhrbXN0ZXN0MFwwDQYJKoZIhvcNAQEBBQADSwAwSAJBAK5XNlSfmnEZ4
qqRfHYeTdqslGDpJ3uQtue7+UZntdyhpeAe0pgGt+NLgQYnHcdkZH8h6VtKtzZv
4mO4fFGNfgsCAwEAAaOCAS4wggEqMAkGA1UdEwQCMAAwgawGA1UdIASBpDCBoTC
BngYLYIZIAYb4RQEHAQEwgY4wKAYIKwYBBQUHAgEWHGh0dHBzOi8vd3d3LnZlcm
lzaWduLmNvbS9DUFMwYgYIKwYBBQUHAgIwVjAVFg5WZXJpU2lnbiwgSW5jLjADA
gEBGj1WZXJpU2lnbidzIENQUyBpbmNvcnAuIGJ5IHJlZmVyZW5jZSBsaWFiLiBs
dGQuIChjKTk3IFZlcmlTaWduMAsGA1UdDwQEAwIFoDARBglghkgBhvhCAQEEBAM
CB4AwTgYDVR0fBEcwRTBDoEGgP4Y9aHR0cDovL3BpbG90b25zaXRlY3JsLnZlcm
lzaWduLmNvbS9PblNpdGVQdWJsaWMvTGF0ZXN0Q1JMLmNybDANBgkqhkiG9w0BA
QQFAANBAIt+n2Y6AKeZIrQlyLR3oZu8dGpM9ad1tUzg1d1uJ4olt0NT4pFrTLbO
2tfqWBBKYQoLaZ+9QYlcZ1joMISLWzw=</dsig:X509Certificate>
                  </dsig:X509Data>
               </KeyInfo>
            </KeyBinding>
         </Answer>
         <dsig:Signature
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
            <dsig:SignedInfo
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
               <dsig:CanonicalizationMethod
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
```

```
               <d:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"
xmlns:d="http://www.w3.org/2000/09/xmldsig#"/>
               <dsig:Reference URI="#refId_0"
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                  <dsig:Transforms
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
                     <d:Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-
signature" xmlns:d="http://www.w3.org/2000/09/xmldsig#"/>
                     <dsig:Transform
Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                  </dsig:Transforms>
                  <dsig:DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>

<dsig:DigestValue>ReUiRRDO8Ss+U80hs7JgURex9rM=</dsig:DigestValu
e>
               </dsig:Reference>
            </dsig:SignedInfo>
            <dsig:SignatureValue
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">NTsEaNecScT0JWx
Bl6F7WizXh8ptDkilkSxeUgI0Obv/qRdeTPReA1LRZaVWg7DPMTxt63UqcIl0hG
e/ezpOfv4leHZ3+pDV2WLYiyxnbpIYVteqDOYGTpbw0Jjb/H3ZrBrhJUqvF4g3e
SYwczBPj8R1JHbgCEz6M/CNoPafY8c=</dsig:SignatureValue>
            <dsig:KeyInfo
xmlns:dsig="http://www.w3.org/2000/09/xmldsig#">
               <d:KeyValue
xmlns:d="http://www.w3.org/2000/09/xmldsig#">
                  <d:RSAKeyValue>

<d:Modulus>ANjD0DM0mcCOoEhYC4X551U6q/78RHwF+cZHmMSYTa3HJKLcmpQ/
XaWpPDH7JQ9VIq8QyAT3x1+4Me/LY9Mt/V4KBb+gEHx9DRpWMY3T85JYIGfvHB1
d4k35RBoFBAK8Dg5kvEpaBL2hqsxPxJeln0KxDeHu/8bKtO7zd9E9Vn3H</d:Mo
dulus>
                     <d:Exponent
xmlns:d="http://www.w3.org/2000/09/xmldsig#">AQAB</d:Exponent>
                  </d:RSAKeyValue>
               </d:KeyValue>
            </dsig:KeyInfo>
         </dsig:Signature>
      </RegisterResult>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```