uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Ying Liu

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.Sc. (Systems Science)

GRADE / DEGREE

Systems Science

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Enhancing Security for XML Web Services

TITRE DE LA THÈSE / TITLE OF THESIS

Professor Tet Yeap

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Professor A. Karmouch

Professor J. Yao

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

ENHANCING SECURITY FOR XML WEB SERVICES

Ying Liu

Thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
In partial fulfillment of the requirements
For the MSc degree in Systems Science

School of Information Technology and Engineering
University of Ottawa

Supervised by Professor Tet Hin Yeap

Your file Votre référence
ISBN: 978-0-494-34084-4
Our file Notre référence
ISBN: 978-0-494-34084-4

Declaration

THE UNDERSIGNED HEREBY DECLARES THAT THE WORK CONTAINED IN THIS THESIS IS MY OWN ORIGINAL WORK AND HAS NOT PREVIOUSLY IN ITS ENTIRETY OR IN PART BEEN SUBMITTED AT ANY UNIVERSITY FOR A DEGREE

YING LIU

JUNE, 2006

Acknowledgement

I would like to express my deep and sincere gratitude to a long list of people who have been supporting me, encouraging me and inspiring me during the time I worked on this thesis project.

- Firstly, I would like to give my deep thanks to Professor Tet Yeap. Thank you for giving me such an opportunity to have this beautiful journey of getting my MSc degree. It was you who sparked my interest in this research. Thank you very much for your encouragement, patience and efforts on this thesis project, as well as your warm and kind support.
- Secondly, I would like to thank my parents and my sisters for their love, support and patience.
- Thirdly, I would like to thank my husband, Zhenxiao Liu, for instilling in me the passion to learn and for his long-lasting understanding and support through the years.
- Finally, I would like to thank my friends and colleagues, especially Monica Williams, for their support and assistance.

Abstract

The XML-based interoperable characteristics make enhancing security for XML Web Services a lot different from that of the traditional network-based applications. SSL VPN gateways are usually used to provide security for traditional network-based applications, but not for Web Services. This thesis presents an integrated security solution for securing both traditional network-based applications and Web Services.

The integrated security solution includes a VPN framework and a Web Services framework. Considering that we have already had an SSL VPN gateway developed by our lab, we take it as the motherboard of the solution and the VPN server of the gateway as the security functional part of the VPN framework. As the highlight of this thesis project, a Web Services security component, also the security functional part of the Web Services framework, has been developed, implemented and integrated with the SSL VPN gateway to get the integrated security solution.

The problem of applying ECC over binary fields for XML security, SOAP message security and Web Services security to make the Web Services security component share the same set of ECC keys with the VPN server on the gateway has been studied. Tools for reading ECC keys and certificates from the BUL's key files have been developed. Methods to adopt the ECC key files to ensure security of Web Services have also been developed. To the best of our knowledge, there is no previous work on adopting ECC keys over binary fields for Web Services security.

The integrated security solution we present in this thesis is the prototype of a network device that has functions of three gateways: a VPN gateway, a Web Services security gateway and a Web Services gateway.

Table of Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

3DES	Triple DES
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
ASN.1	Abstract Syntax Notation One
BUL	Bell University Laboratories
CA	Certificate Authority
CPU	Central Processing Unit
DES	Data Encryption Standard
DL	Discrete Logarithm
DSA	Digital Signature Algorithm
ebXML	Electronic Business using eXtensible Markup Language
EC	Elliptic Curve
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman Key Agreement Method
ECDSA	Elliptic Curve Digital Signature Algorithm
ECIES	Elliptic Curve Integrated Encryption Scheme
FTP	File Transfer Protocol
HMAC	keyed-Hash Message Authentication Code
HTML	HyperText Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	HTTP over SSL
IEEE	Institute for Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
JDK	Java Development Kit
JRE	Java Runtime Environment
KDF	Key Derivation Function
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Code

NIST	National Institute of Standards and Technology
OASIS	Organization for the Advancement of Structured Information Standards
PEM	Privacy Enhanced Mail
PKI	Public Key Infrastructure
RSA	Ron Rivest, Adi Shamir, and Leonard Adleman Algorithm
SAML	Security Assertion Markup Language
SEC	Standards for Efficient Cryptography Group
SGML	Standard Generalized Markup Language
SHA	Secure Hash Algorithm
SMB	Small and Medium Business
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Socket Layer
TLS	Transport Layer Security
UDDI	Universal Description, Discovery, and Integration
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WSDD	Web Service Deployment Descriptor
WSDL	Web Service Description Language
WS-I	Web Services Interoperability Organization
WSS4J	Web Services Security For Java
WWW	World Wide Web
XACML	eXtensible Access Control Markup Language
XKMS	XML Key Management Services
XML	Extensible Markup Language

Chapter 1 Introduction

1.1 Problem statement

Web Services technologies have been in their popularity for a while. Since the concept of Web Services was prompted in 1999 for the first time, Web Services technologies have become almost well-fledged. Now more and more enterprises and e-business companies have developed and deployed or have begun to develop and deploy Web Services. Web Services are believed to be the next step of Internet-based applications and transactions.

By utilizing XML, SOAP, HTTP, WSDL, UDDI protocols and standards, Web Services technologies provide us with a new way of unifying disparate systems into an openly interoperable environment. The XML-based characteristics of Web Services make Web Services available via the Web no matter how different their background systems are between the service provider and the service consumer. Thus the Web Services technologies are widely used in the areas of distributed computing and systems integration.

Also the Web Services' XML-based interoperable characteristics create a lot of challenges to enforcing the security issues for Web Services over the Internet. Although authentication, authorization, confidentiality, integrity and non-repudiation are still the five security issues should be addressed when securing Web Services, the unique characteristics of Web Services make the traditional security solutions such as SSL/TLS are not enough. Web Services need message level security.

To meet the requirements for Web Services Security, many efforts have been made. As technologies on Web Services security are maturing, products on Web Services security come to the markets. XML encryption and XML digital signature, as the two basics among Web Services security technologies, are always implemented by Web Services security gateways, either as hardware devices or software architectures, to provide message level security to Web Services. However, these gateways usually do not provide

security for the traditional network-based applications that are not developed with Web Services standards and technologies.

On the other hand, VPN technology, as one of the traditional security solutions, usually uses VPN gateways as its entry points to enforce the security policies for securing traditional network-based applications. The VPN gateways secure accesses to the applications behind gateway over insecure networks such as the Internet. However, these gateways are not aware of Web Services and can not provide message level security for them.

Is it possible to get an integrated security solution for securing both the traditional network-based applications and Web Services by integrating the functions of VPN gateways and the functions of Web Services Security gateways?

1.2 Motivation and objectives

Based on the above problem statement, we are led to the idea of providing an integrated security solution for securing both traditional network-based applications and Web Services by integrating the functions of VPN gateways and the functions of Web Services Security gateways.

An SSL VPN gateway has already been developed by Bell University Labs (BUL) at the University of Ottawa. It has been proven that the gateway works efficiently for securing the traditional network-based applications, though the security policies are simple now. Utilizing ECC keys to enforce its cryptography-related security policies is one of this VPN gateway's characteristics.

Considering that we have already had the box of the VPN gateway and the control policies and key file used by the VPN server to secure the network-based applications, what we need to do to get the integrated security solution are,

- to propose the integrated security solution using the SSL VPN gateway as the motherboard and the VPN server as the security functional part for securing the traditional network-based applications.

- to develop and implement an XML Web Services security component as the security functional part for securing Web Services.
- to integrate the Web Services security component with the VPN server on the VPN gateway.

With this work done, the integrated box can efficiently secure both the traditional network-based applications and Web Services. The control policies and key file of the VPN server can be shared by the XML Web Services security component. With little cost, compared to other security products, this integrated box is suitable for Small and Medium Business (SMB) who would like to apply security measures to both the traditional network-based applications and Web Services.

The integrated box will be the enterprise's entry point in the Internet. When the requests for the traditional applications come, they are processed by the VPN engine. When the requests for Web Services come, they are processed by the Web Services security component. Moreover, the Web Services security component can also treat the requests differently according to where the requests originate, inside or outside the VPN.

Sharing the same control policies and key file by the BUL's VPN server and the XML Web Services security component is part of the integration work. The BUL's ECC keys used by the BUL'S VPN server will be one of the unique characteristics of the Web Services security component and the integrated box as well. Thus,

- to apply the BUL's ECC keys to the Web Services security component makes another goal of this thesis project

1.3 Contributions

The motivation and objectives above actually envisioned the contributions of this thesis project. These contributions are listed as follows.

1. An integrated security solution for securing both traditional network-based applications and Web Services is proposed based on an SSL VPN gateway that we have already had for securing the traditional network-based

applications. Two frameworks are included in this solution. One is the VPN framework and the Web Services framework.

2. An XML Web Services security component for securing Web Services is developed, implemented and integrated with the VPN server on the gateway. Contributions of this work are listed as follows.

3. Other components, which together with the Web Services security component compose the Web Services framework in the integrated security solution, are developed. They are Web Services clients, Web Services on the gateway, Web Services on the internal application server and a Web Service proxy. These components compose the Web Services framework of the solution.

4. Tools for reading BUL ECC keys and certificates from PEM files are developed.

5. Encryption algorithm ECEIS and signature algorithm ECDSA are studied. Support for these two algorithms with BUL ECC keys is added in the jBorzoi library.

6. Methods to support XML signing/verifying and encrypting/decrypting with BUL ECC keys are studied and developed using the Apache XML Security library.

7. Methods to support SOAP message signing/verifying and encrypting/decrypting with BUL ECC keys are studied and developed using the Apache WSS4J library.

8. With further development, the integrated security solution will become a network device with functions of three separate network devices, the VPN gateway, the Web Services security gateway and the Web Services gateway. It can be applied by SMB as an efficient way of securing both traditional network-based applications and Web Services and mapping

Web Services and legacy applications in the internal network to Web Services on the gateway.

The work of this thesis project was done by Java technology mainly with related libraries from the Apache Software Foundation. In addition, to the best of our knowledge, there is no previous work on adopting ECC keys over binary fields for Web Services security.

1.4 Outline of this thesis

This thesis is organized as follows.

Chapter 2 is an introduction to the background and related research work of this thesis project. The concepts of XML and the Web Services architecture are reviewed first in this chapter followed by mechanisms of Web Services security. Security measures such as SSL/TLS and firewalls are not enough, as there are new challenges to Web Services security. Some Java projects related to Web Services and Web Services security are presented. Lastly, two commercially available XML security products are analyzed in this chapter.

Chapter 3 proposes the integrated security solution. The overview of the security solution is given first, followed by the advantages of this solution. Next, the functionalities of the Web Services security component and the Web Services proxy are described. The integration of the Web Services security component with the VPN server is described at the conclusion of this chapter.

Chapter 4 is on the design issues of the integrated security solution. The functions and architecture of Web Services clients, Web Services on the gateway, Web Services on the internal application server, the Web Services security component and the Web Services proxy are described.

Chapter 5 is on the design principles of the Web Services security component. We describe in depth the elliptic curve cryptography and its application in XML security, SOAP message security and Web Services security. The elliptic curve cryptographic schemes are used in this thesis to implement our Web Services security component integrated with the VPN server, both using the same set of elliptic curve cryptographic

keys. XML security method and syntax, including XML encryption and XML digital signature, are analyzed before analysis of the SOAP security method and syntax. The problem of Web Services security is studied at the end of this chapter.

Chapter 6 is dedicated to the implementation of the XML Web Services security component and setting up of the related systems. First, implementation of each module in the Web Services framework is described. Then the algorithms and mechanisms of reading ECC keys, ECC encryption and signature, encrypting/decrypting XML/SOAP messages and signing/verifying digital signatures in XML/SOAP messages are given in detail. The running results are given at the end of this chapter.

We conclude our thesis and give future direction of this thesis in Chapter 7.

Chapter 2 Web Services and Web Services Security

This chapter gives an overview of the research works related to this thesis project. Technologies concerning XML, Web Services and Web Services security will be introduced first, followed by an introduction to the Web Services security gateways.

2.1 XML

Extensible Markup Language (XML) [1] is a simple data description language. It was standardized by World Wide Web Consortium (W3C) to bring a way of markup-based publishing to the Web.

XML is a subset of SGML and similar to HTML in format. However, it is different from HTML. XML is a well-formed markup language and does not have a specified tag set. XML's tag set is open. XML has no grammar requirement for the language either. You can define your own tags to describe the data in any way you want, as long as it conforms to the general structure that XML requires.

The XML document is plain text. It is platform-independent and programming language neutral. Any application can understand it. So besides its support for human access to information, more important is that it enables a flexible way of describing data in a machine-processable manner. Since its emergence, it has been profoundly affecting the way we build and think about Web and Web Services. It is the foundation of Web Services. This is the reason why XML is closely related to Web Services. When we are talking about Web Services, we always mean XML Web Services.

2.2 Web Services

XML Web Services, as a technology, are a set of protocols that build on the global connectivity made possible by SOAP, XML and HTTP [5]. The main objective of the XML Web Services is to provide interoperability to applications distributed on networks

including the Internet. The Web Services expose their interfaces so that a user can build a client application to communicate with the Web Services.

The standards and protocols, XML, SOAP, HTTP, WSDL (Web Service Description Language), UDDI (Universal Description, Discovery and Integration), etc., compose the so-called XML Web Services technology stack. In this XML Web Services technology stack, XML is the foundation, because XML itself is an open standards; most of the other open standards and protocols such as SOAP, WSDL and UDDI are all XML-based. This is the greatest of the Web Services, which makes the Web Services universally accessible and thus makes the Web Services technology a great success.

Typically, Web Services use WSDL to describe their interfaces, use UDDI as the yellow and white pages to publish their interfaces to the public and use SOAP to communicate with the service user, which could be a person, an application or a Web Service.

A typical simple Web Services architecture is illustrated as follows, along with a description of the protocols and standards in this architecture.

2.2.1 Web Services architecture

The architecture is depicted in Figure 1. In this architecture, the Web Services provider publishes interfaces to Web Services to a repository and the Web Services requester queries the repository for the interfaces. The Web Services requester then calls Web Services with a Web Services client.

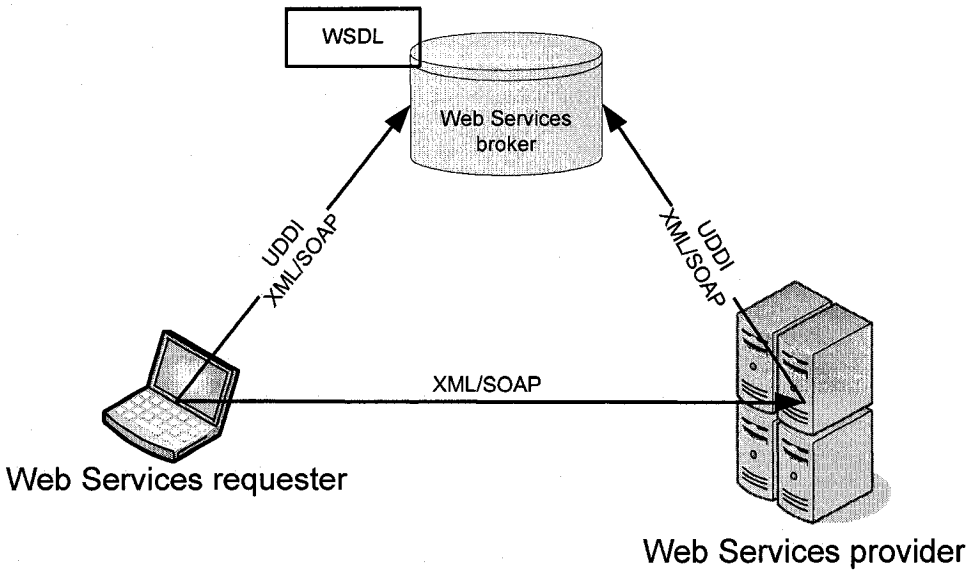

Figure 1 Architecture of Web Services

2.2.2 SOAP

SOAP [16] is a protocol for exchanging information in a decentralized, distributed manner. SOAP provides the means of communicating for XML Web services. It is a W3C specification.

The structure of a SOAP message is depicted in Figure 2. A SOAP message is composed by an envelope containing the body of the message and the necessary header information used to describe the message. A SOAP message is itself an XML document conforming to specific XML schema.

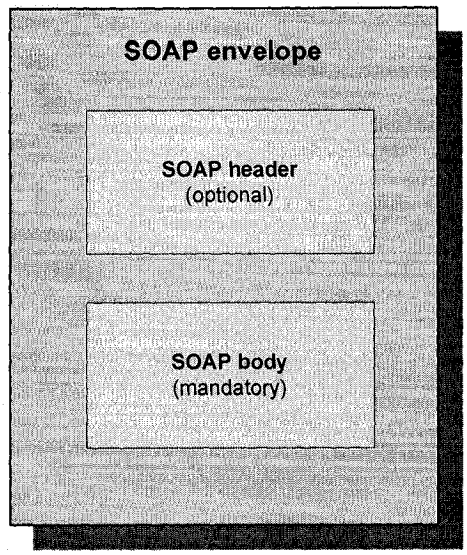

Figure 2 A SOAP envelope

SOAP messages can be sent over communication protocols such as FTP, HTTP and SMTP; however, the usual way is SOAP over HTTP, which is depicted in Figure 3. In this scenario, communication between the SOAP client and the SOAP server is through the underlying HTTP client and HTTP server.

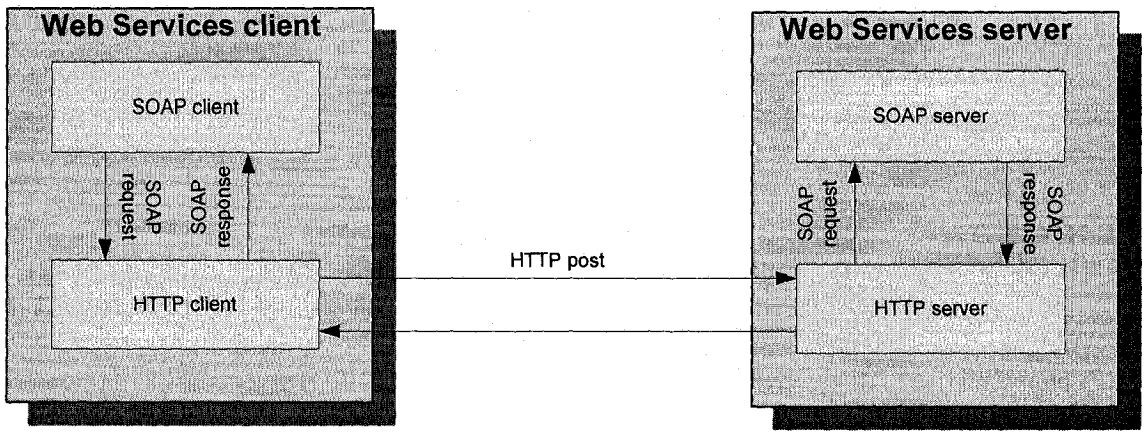

Figure 3 SOAP over HTTP

SOAP is the core technology of Web Services. WSDL and UDDI are all based on SOAP to transfer information related to Web Services description and publishing.

SOAP messages can transverse more than one hop and it can be over different communication protocols on each hop. For example, it can be sent over FTP on the first

hop, over HTTP on the second hop and sent over email by SMTP protocol on the third hop. This is the reason that security measures are usually applied on SOAP messages instead of the communication protocols.

As SOAP message is itself based on XML, SOAP message security is also based on XML security. Processing SOAP message security is the majority of the work of the Web Services security component proposed in this thesis.

2.2.3 UDDI

UDDI [17] provides a central directory service for publishing technical information about Web Services. It allows businesses to register with the directory service so they can advertise their services; thus they can find each other's services and carry out transactions through them on the Web.

UDDI is a Web directory usually built over SOAP, and UDDI itself is a Web Service. UDDI over SOAP, which is over HTTP in turn, is depicted in Figure 4. In this scenario, the UDDI client sends queries to the UDDI server on a Web Services broker node. The UDDI server then queries the Web Services registry data through the UDDI processing engine and returns information about Web Services such as programmatic interface. The messages between the UDDI client and the UDDI server are also sent over SOAP, which is over HTTP in turn.

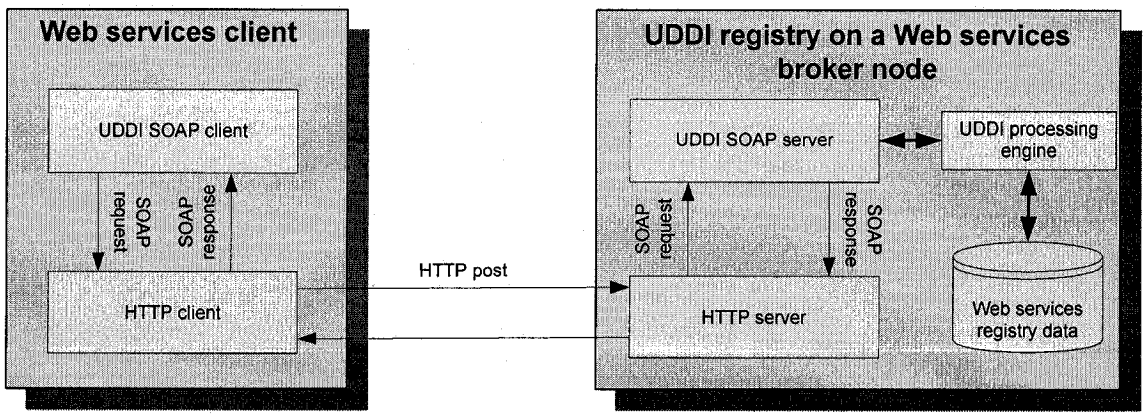

Figure 4 UDDI over SOAP

One thing to notice is that Web Services do not have to be published and found by UDDI. In a private Web Services deployment environment, within a VPN, for example, the interfaces to Web Services can be sent via email or published by files. Of course, this can also be done by UDDI; however, it is not a cost-effective way when there are not so many Web Services and participating parties.

2.2.4 WSDL

WSDL [18] is an XML-based language layered on top of the schema that describes a Web Service. It provides the necessary information for the Web Services client to interact with the Web Services provider.

WSDL specifies the information and its style in Web Services request and response messages. A part of a sample WSDL file is shown in Listing 1. The name and location of the Web Service are specified in the sample WSDL file.

```
  <wsdl:message name="showBULInfoResponse">
    <wsdl:part element="impl:showBULInfoReturn" name="showBULInfoReturn"/>
  </wsdl:message>
  <wsdl:message name="showBULInfoRequest">
    <wsdl:part element="tns1:in0" name="in0"/>
  </wsdl:message>
  <wsdl:portType name="ServiceGateway">
    <wsdl:operation name="showBULInfo" parameterOrder="in0">
      <wsdl:input message="impl:showBULInfoRequest" name="showBULInfoRequest"/>
      <wsdl:output message="impl:showBULInfoResponse" name="showBULInfoResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="service-gatewaySoapBinding" type="impl:ServiceGateway">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/
http"/>
    <wsdl:operation name="showBULInfo">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="showBULInfoRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="showBULInfoResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="ServiceGatewayService">
    <wsdl:port binding="impl:service-gatewaySoapBinding" name="service-gateway">
      <wsdlsoap:address location="http://localhost:8080/axis/services/service-
gateway"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Listing 1 WSDL sample

As WSDL is XML-based, it is also extensible. It can be extended to describe any Web Services. WSDL is generally produced by applications and read by applications. Transferring WSDL between applications is through SOAP.

The WSDL description published by Web Services can be used to generate routines to simplify programming of Web Services clients as we do in Chapter 6. In this way, we do not have rewrite the Web Services client for every Web Service, but only create the part for locating Web Services.

2.3 Web Services security

Web Services security involves a lot of technologies, including standard HTTP security technologies for transport security and XML security technologies. It also includes security technologies such as that at the level of operating systems, which is not covered in this thesis.

2.3.1 Challenges to Web Services security

For Web Services security, traditional security technologies are usually not working. For traditional applications on the Internet, firewalls are powerful network devices to ensure security. They provide intrusion detection and prevention for various applications. Usually firewalls are and have to be set up to let HTTP traffic through to provide access to Web applications. As in most cases where Web Services requests and responses are utilizing HTTP protocol, traditional firewalls cannot secure Web Services.

Most of the traditional applications over the Internet do not require semantic processing of the messages in transition, so some security technologies such as SSL/TLS are good enough for them. However, this is not true for Web Services. Web Services require processing at the message level on the intermediate nodes of message transition. It makes the traditional security technologies inadequate.

Another challenge to Web Services security is the interoperability between security technologies from different manufactures for different applications. Security technologies for other applications were never designed to meet interoperability as standards for Web Services. Web Services and Web Services security technologies are XML-based, and it is

never a big issue for different security technologies to cooperate, as there are universal open standards for Web Services.

The XML security framework presented in the next section addresses these challenges.

2.3.2 XML security technologies

The XML security framework recommended by W3C includes three technologies: XML Encryption, XML Digital Signature and XML Key Management Services.

1) XML Encryption

XML Encryption supports the encryption of all or part of an XML document. XML Encryption is not locked into any specific encryption scheme. XML Encryption can encrypt the entire XML document, an element and all its subelements, the content of an XML element or a reference to a resource outside the document.

More information about XML Encryption can be found in [19].

2) XML Digital Signature

The specification of XML Digital Signature defines the syntax and rules for processing XML digital signatures that provide integrity, message authentication and signer authentication services for data.

More information about XML Digital Signature can be found in [20].

3) XML Key Management Services (XKMS)

XKMS ensures trust processing to one or more specialized trust processors. It gives businesses an easier way to manage digital signatures and data encryption. With XKMS, all the participating institutions of XML Web Services can use standard interfaces to work with each other. One of the goals of XKMS is to make it easier for Web Services clients by offloading the complexity of processing PKI to some external services. The difference between systems using XKMS and not using XKMS is depicted in Figure 5.

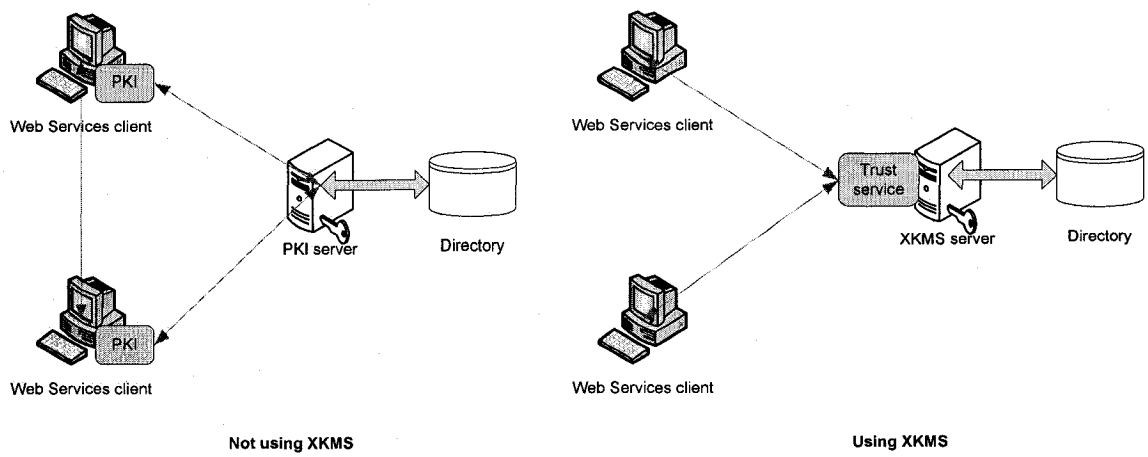

Figure 5 Systems using and not using XKMS

When not using XKMS, each Web Services client has to provide PKI functionalities itself. Implementing PKI is always very complex and has to be done on every Web Services client. The PKI component on the Web Services clients interacts with a PKI server, which is connected to public key directories.

With the XKMS infrastructure, Web Services clients with interact with an XKMS server through standard Web Services interface, usually implemented with SOAP. The XKMS server is connected to public key directories. Participating parties, including Web Services clients and Web Services servers, interact with the XKMS server as authentication proxy.

We do not use XKMS in this thesis, as it is not very closely related to the Web Services security component. Another reason is that we want to develop the Web Services security component with the existing PKI infrastructure used by the BUL's VPN gateway. Implementing XKMS working with the Web Services security component is left to future research.

More information about XKMS can be found in [21].

2.3.3 Activities on Web Services and Web Services security

There are many activities involving Web Services security.

World Wide Web Consortium (W3C) is an international consortium for developing standards for the World Wide Web. W3C was created to ensure compatibility and agreement among industry members by adopting new standards. W3C defined XML, XML-Schema, SOAP, WSDL and Web Services architectures. WSDL 1.1 has not been endorsed by W3C, but WSDL 2.0 is the W3C recommendation.

The Organization for the Advancement of Structured Information Standards (OASIS) is "a not-for-profit, international consortium that drives the development, convergence and adoption of e-business standards." Activities sponsored by OASIS involving Web Services include SAML, XACML and WS-Security. SAML is an XML standard for exchanging authentication and authorization information between an identity provider and a service provider. XACML (eXtensible Access Control Markup Language) is an XML standard for access control that specifies how authorizations for XML documents should be designed based on rules, policies and policy sets.

Web Services Interoperability Organization (WS-I) is "an open industry organization chartered to promote Web Services interoperability across platforms, operating systems and programming languages." WS-I does not develop standards, but endorses the use of existing standards. The WS-I Basic Profile provides interoperability guidance for core Web Services specifications, including SOAP, WSDL and UDDI. WS-I Basic Security Profile is designed to address interoperability across transport security, SOAP, messaging security and other security considerations for the WS-I Basic Profile. The WS-I Basic Security Profile is targeted at HTTP over TLS and Web Services security through SOAP message security.

Electronic Business using eXtensible Markup Language (ebXML) is a joint activity of OASIS and the United Nations Center For Trade Facilitation and Electronic Business (UN/CEFACT). ebXML is to provide guidelines for doing electronic business in an interoperable, secure and consistent way. Web Services technologies such as SOAP, WSDL and UDDI can be used with a specification of ebXML.

2.3.4 Apache projects related to Web Services and Web Services security

1) Tomcat

Apache Tomcat is a servlet container that implements the Java Servlet and the JavaServer Pages specifications from Sun. It provides an environment for running Java code in cooperation with a Web server.

Tomcat can be used as a servlet engine in cooperation with another Web server such as Apache Web server. In this case, Tomcat is in charge of the dynamic Web contents that are determined by servlets, and the Web server is in charge of the static Web pages with certain directory structure. Connectors are used between the Web server and Tomcat to direct certain traffic to Tomcat for future process.

Tomcat can also be used as a Web server itself as in this thesis when the static contents are not meant to be a heavy burden and when the structure of Web site is not very complex.

A lot of industries and organizations adopt Tomcat to support their various Web applications [53].

2) Axis

Apache Axis is an implementation of the SOAP specification submitted to W3C. The Apache Axis project is a follow up to the Apache SOAP project. It is an open source XML-based Web Services development environment. There are two versions of Axis; one is done with Java, and the other is done with C++.

In Axis, there are two ways of exposing Web Services implemented with Java code. Users can deploy the Java source code as Java Web Services or deploy compiled Java classes with a Web Service Deployment Descriptor (WSDD) file.

Axis provides a mechanism to support SOAP message handling at the server side and the client side. It provides a message context where users can insert operations on SOAP messages. The Web Services security project, Apache WSS4J, is based on the Axis project to apply Web Services security measures to SOAP messages.

Many companies and individuals around the world use Axis to support Java Web Services.

3) WSS4J

Apache WSS4J is an implementation of the OASIS Web Services Security (WS-Security) specification.

WSS4J provides mechanisms that developers of Web Services can use to encrypt/decrypt the SOAP messages and sign/verify digital signatures applied on SOAP messages. It requires Apache Axis for SOAP message processing and Apache XML Security for encrypting/decrypting XML messages and signing/verifying digital signatures in XML messages.

4) XML Security

Apache XML Security library is an implementation of the W3C recommendation on XML Encryption [40] and the W3C recommendation on XML Signature [41].

XML Security provides APIs for XML processing, including XML encryption and XML digital signature, as a basis for XML-based security including SOAP message security. There are two versions of the XML Security library; one is done with Java, and the other is built with C++.

2.3.5 Other Java projects

Besides the Apache projects, there are some Java projects from other institutes for Web Services and Web Services security. These projects include the Dragongate jBorzoi project and the BouncyCastle project.

1) Dragongate jBorzoi

The Dragongate jBorzoi project works on the elliptic curve cryptography library. jBorzoi is the Java version and Borzoi is the C++ version, and both are open source.

jBorzoi provides support for the Elliptic Curve Integrated Encryption Scheme (ECIES), the Elliptic Curve Digital Signature Algorithm (ECDSA) and the Elliptic Curve Diffie-Hellman Key Agreement Scheme (ECDH) for binary field elliptic curves. It supports elliptic curves with several elliptic curve domain parameter recommendations from NIST.

2) BouncyCastle

BouncyCastle, a free cryptographic API, is developed by the Legion of the Bouncy Castle. This cryptographic API can be used anywhere there is a restriction against exporting strong Java cryptographic algorithms.

It supports the processing of X.509 certificates. It also supports elliptic curve public key cryptography, but at this time, only the ECIES algorithm and ECDSA algorithm over the prime field are supported.

2.4 Web Services security gateways

Web Services security in practice is usually implemented by the Web Services security gateway as a standalone network device. The role of Web Services security gateways protecting XML Web services is much like the role of traditional firewalls protecting traditional network-based applications. The security gateways should work with other network devices to work properly and effectively. Role of the Web Services security gateway is depicted in Figure 6.

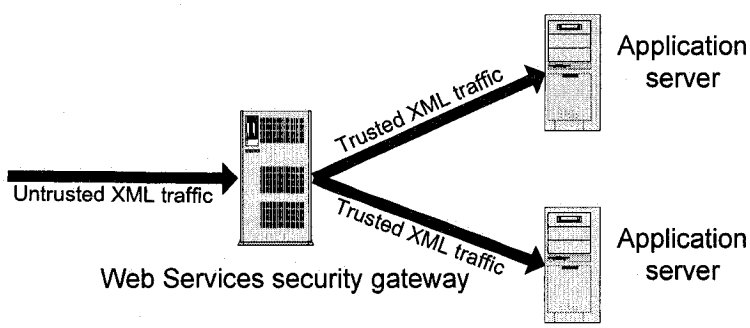

Figure 6 Role of the Web Services security gateway

Upon receiving XML Web Services traffic, the Web Services security gateway analyzes the XML traffic and determines whether the traffic contains malicious content and meets other security criteria. The Web Services security blocks XML traffic that will harm the application server or does not meet the security criteria. In this way, the Web Services security gateways work as an XML traffic filter.

Institutes may choose to apply security measures on each of their application servers on an entry point, the Web Services security gateways. Similar to what is done for the

traditional network-based applications, applying security measures on the Web Services security gateways is preferred, despite some minor disadvantages.

Building a shared security gateway for Web Services on internal application servers has the following advantages.

- Higher performance and lower cost. Maintenance of the hardware and software is at lower cost compared to maintenance on every application server. By deploying the Web Services security gateways, enterprises can reduce the total cost of ownership by moving processing from application servers to a shared entry point. In this architecture, there are only centralized patching and updating of Web Services security gateways and a centralized process for data validation, which is cost-effective. Configuration of security policies can be done more easily, as it only needs to be done on the security gateway instead of on every application server.
- A centralized and consistent security policy. It is easy and desired to have uniform security policies for all the application servers. There would not be conflicts of security policies among different application servers.
- The security gateway can be integrated with other network devices such as the traditional gateway or the VPN gateway as a software module, as done in this thesis; for better cooperation between the Web Services security component and the original gateway software.
- Less entry points, less threat and damage. In an institute where only Web Services gateways are exposed to the Internet, the internal network and application servers can be masked with the proper configuration.

2.4.1 Commercial implementations of Web Services security gateways

There are several vendors that present XML Web Services security gateways and claim that the XML security gateways work great for XML Web services, including that from Datapower and Forum.

In our opinion, however, as Web Services development and deployment are still at the starting point, one has to be careful when investing in large-scale Web Services security gateways. Moreover, we have to be prepared for the future changes to network architecture.

1) Datapower XML security gateway

Datapower XS40 is an XML security gateway product that is widely quoted on the Web. Most of its parts are implemented as hardware to keep it processing XML traffic at wire-speed [7].

This product is used by some institutes to secure their Web Services. It is claimed to be fast for Web Services processing. This is for large enterprises to secure their Web Services. But for an SMB, this will be too expensive to purchase and maintain these devices.

2) Forum XWALL XML security gateway

The Forum XWALL is said by its vendor to be "the industry's first Web Services Firewall equipped with data authentication as well as XML intrusion prevention to actively protect against XML viruses, data corruption and denial of Web service attacks."

The Forum XWALL Web Services security gateways are also for large enterprises and will be too expensive for SMB.

2.4.2 Standalone Web Services security gateways versus integrated security solutions

XML Web Services security gateways are the latest solution for XML Web services. They stand in face of malformed incoming XML traffic and process the XML traffic before it comes to the application server residing inside of enterprises.

However, there are arguments on the Internet saying that XML security gateways may disappear or be integrated into other network devices such as routers or firewalls in the next few years. Today's Web Services security gateways are the result of the fact that development and deployment of Web Services are still at the starting point and there are no standard and mature solutions for Web Services security. When the Web Services

security solutions are mature and securing Web Services is a must-have feature of network security device, there will be integrated solutions.

Using a standalone network device for Web Services security also has the problem of complex network architecture. There will be more than one entry point to the internal network. Applying security policies on these entry points will be more complex than applying security policies on just one entry point.

2.5 Chapter summary

Web Services architecture and Web Services security were briefly reviewed in this chapter.

Web Services are based on XML, and so is the Web Services architecture. SOAP is the protocol for transferring Web Services traffic that is based on XML. UDDI and WSDL are all based on SOAP.

Web Services security is closely related to its XML characteristics. XML encryption and XML digital signature are two basic technologies for Web Services security. The Web Services security gateways are used in practice to security Web Services. Web Services security functions will be integrated into other network devices.

Chapter 3 The Proposed Integrated Security Solution

The integrated security solution proposed in this thesis is for securing both traditional network-based applications and Web Services, using the same set of ECC keys over binary fields from the BUL. For these two types of applications, this integrated security solution includes a VPN framework and a Web Services framework.

The overview of this integrated security solution is given first, followed by description of the VPN framework and the Web Services framework. How these two frameworks are integrated is given next. We discuss the advantages of this integrated security solution at the end of this chapter.

3.1 Solution overview

The proposed integrated security solution is shown in Figure 7.

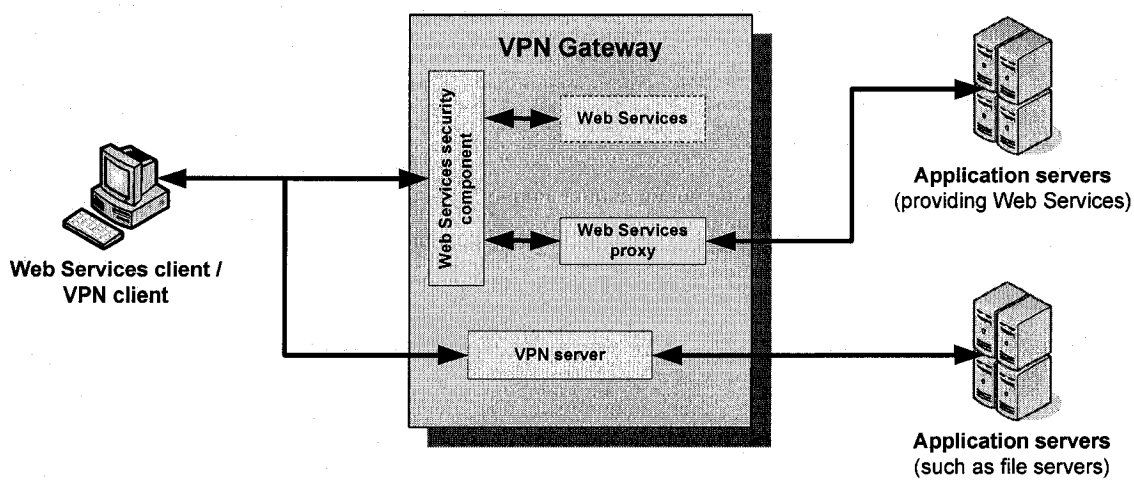

Figure 7 Architecture of the integrated security solution

The integrated security solution includes three major components on the gateway: a VPN server, a Web Services security component and a Web Services proxy. This is the

prototype of a network device that has the functions of three network devices: a VPN gateway, a Web Services security gateway and a Web Services gateway.

This integrated security solution also includes some other components as the running environment of the three major components. So the overall integrated security solution includes a Web Services on the gateway and the corresponding Web Services client, a Web Services on the internal application server and the corresponding Web Services client, a Web Services proxy mapping Web Services on the internal application servers to Web Services on the gateway, the Web Services security component, a VPN server and the corresponding VPN client. The VPN client and VPN server compose the VPN framework. All the other components compose the Web Services framework.

In this integrated security solution, the VPN server in the VPN framework and the Web Services security component in the Web Services framework use the same key file. However, a user may connect to VPN using the VPN client with a key file and then request Web Services using the Web Services client with another key file.

As seen in Figure 1, there are three participating parties in a real Web Services architecture. The Web Services framework in our proposed architecture does not include the UDDI of the Web Services architecture. However, for UDDI to authenticate and authorize the Web Services providers and Web Services requesters, the Web Services security component can also used.

3.2 The VPN framework

The VPN framework provides functions of a VPN gateway. It is for securing traditional network-based applications. In this framework, communication between VPN clients and application servers in the internal network behind the gateway is secured by the VPN server.

3.2.1 VPN server

The VPN server sets up a VPN connections between VPN clients and VPN server to provide security for traditional network-based applications such as domain login and file

sharing. This is an SSL VPN server that utilizes ECC keys over binary fields from BUL. The key file is shared by the VPN server and the Web Services security component.

3.2.2 VPN clients

When using VPN clients to connect to secured resources behind the VPN gateway, there will be a virtual IP address corresponding to each VPN client. By the virtual IP addresses, the VPN server can tell whether a connection is from outside or inside the VPN.

The Web Services security component can also use the information of the virtual IP addresses. Based on this information, the Web Services security component can tell where the requests for Web Services come from, outside or inside the VPN, and then can treat them differently.

3.2.3 Application servers in the internal network

The application servers providing traditional network-based applications are secured by the VPN framework. Secure communication between the clients and the application servers is via the VPN server. The connection to the VPN server can only be made through the VPN clients with proper key files.

3.3 The Web Services framework

3.3.1 Web Services clients

The Web Services clients act as a requester for Web Services on the VPN gateway or on the application servers behind the VPN gateway.

3.3.2 Web Services on the gateway

Web Services are provided on the gateway and are requested by the Web Services clients. No complex logic is implemented by these Web Services, as they are not the majority of the work of this thesis.

3.3.3 Web Services on the internal application servers

Web Services on the internal application servers are also requested by Web Services clients. The difference between these Web Services and Web Services on the gateway is that these Web Services cannot be called directly by the Web Services clients because the internal network is usually not visible to the Web Services clients. Web Services clients calling Web Services on the internal application servers are performed through the Web Services proxy.

VPN clients are not allowed to access internal application servers that provide Web Services. If so, Web Services will by no means be protected at the message level. The related access control rules can be defined on the VPN server.

Usually, users connecting with VPN clients are assigned higher priority than users connecting from public network. When users connecting with VPN clients request Web Services in the internal network, they request through the Web Services proxy as other users do. The Web Services proxy may assign higher priority to these users because they request Web Services from the inside of the VPN.

3.3.4 Web Services proxy

The Web Services proxy is a relay between Web Services clients and Web Services on the application servers in the internal network behind the gateway. Of course, this function unit can also be used to connect Web Services clients and application servers in the internal network that do not provide Web Services. In this case, the Web Services proxy provides a means of supporting legacy applications by transforming them to Web Services. This is similar to some Web Services gateways in a real deployment environment that support legacy application servers.

3.3.5 Web Services security component

This is the majority of the work of this thesis project. The Web Services security component is in charge of Web Services security.

For any requests from Web Services clients requesting Web Services on the gateway, the SOAP messages will be processed by this component. The signature contained in the SOAP messages is checked to implement identification and authentication, integrity and non-repudiation. Authorization is based on the result of identification and authentication. After checking the signature, the SOAP messages are decrypted to get the original messages, which are usually requests for Web Services.

Once Web Services are called and the replies are ready, this component will apply security measures to the reply SOAP messages. Signatures are applied to the SOAP messages using the ECC keys of the gateway. SOAP messages are also encrypted to ensure confidentiality of communication between Web Services clients and the Web Services security component.

Besides signing/verifying and encrypting/decrypting SOAP messages, other security measures such as access control and logging/auditing are also implemented for better integration with the VPN server.

3.4 Integration

In our integrated security solution, the Web Services security component is integrated with the VPN server.

The Web Services security component uses the same key file used by the VPN server. By utilizing the same key file, the network device with the Web Services security component and the VPN server has only one identity represented by the key file. Having one identity is preferred for identity management such as certificate issuing and certificate revocation.

Logging/auditing of Web Services should include information related to the VPN. For example, if the Web Services requests come from within the VPN, the log should include the VPN virtual IP of the Web Services client.

Access control rules should be similar to that of the VPN server. For example, users with certain certificates can be banned from accessing both the VPN and Web Services. From the VPN server, Web Services can know whether the requests for Web Services come

from within the VPN. The Web Services security unit can apply more strict measures if the requests are from outside the VPN.

The integration of the Web Services security component and the VPN server in this thesis is easier than where these two components are on different network devices.

3.5 Advantages of this integrated security solution

The following are some reasons why we propose this integrated security solution.

- Functions of three network devices

Besides the VPN server and the Web Services security component, there is also a prototype of the Web Services gateway in this integrated security solution, the Web Services proxy. With future development, this integrated security solution will also work as a fully functional Web Services gateway.

- Securing both traditional network-based applications and Web Services

With the integrated security solution, access to traditional network-based applications is secured by the VPN server, and access to Web Services is secured by the Web Services security component. Integrating both security components onto one device also reduces the complexity of the network architecture and the cost for network devices.

- Sharing the security policies

By residing on the same network device with the VPN server, a lot of security measures applied on the VPN server can be easily applied on the Web Services security component. This increases the deployment efficiency and eliminates the chance of policy mismatching between different network devices.

- Sharing the same key file

By doing this, the gateway is represented by only one set of keys. The highlight of this thesis project is adopting ECC keys from BUL for the purpose of Web services security, so the Web Services component and the VPN component are both controlled by the Certificate Authority.

3.6 Chapter summary

An integrated security solution for securing both traditional network-based applications and Web Services was proposed in this chapter.

This solution includes two frameworks, a VPN framework and a Web Services framework. The VPN framework is for securing the traditional network-based applications in the internal network. The Web Services framework is for securing Web Services on the gateway and in the internal network. These two frameworks were integrated together by using the same certificate file and similar security policies.

This solution has the functions of a VPN server, a Web Services gateway and a Web Services security gateway.

Chapter 4 Design of the Integrated Security Solution

This chapter is on the design of the integrated security solution, which includes a VPN framework and a Web Services framework. However, our focus is on the design of the Web Services framework because the VPN framework has already been developed there in the box, which we call the VPN gateway.

4.1 Overview

Our integrated security solution includes modules on the client node, on the gateway and in the internal network. Relationship between these modules is depicted in Figure 8.

There are four modules on the client node, i.e., a VPN client, a Web Services client for the Web Service on the gateway, a Web Services client for the Web Service in the internal network and a Web Services security component for securing Web Services requests and responses to and from the gateway. The VPN client and the Web Services security component share the same key file.

On the gateway, there are four modules. The VPN server controls access to the traditional network-based applications in the internal network. The Web Service is corresponding to a Web Services client on the client node. The Web Services proxy, which is also a Web Service, is corresponding to the other Web Services client on the client node. Web Services in the internal network are accessed through the Web Services proxy. The Web Services security component applies security measures such as encrypting/decrypting, signing/verifying, logging/auditing and access control to the Web Service and the Web Services proxy. The VPN server and the Web Services security component share the same key file and log files.

In the internal network, there are two types of application servers. The Web Services application servers provide Web Services to Web Services clients through the Web

Services proxy on the gateway. The traditional application servers provide applications to the VPN clients through the VPN server.

4.2 Data flows in the integrated security solution

The data flows are shown in Figure 8. Message flows are indicated by bold (solid or dotted) arrows.

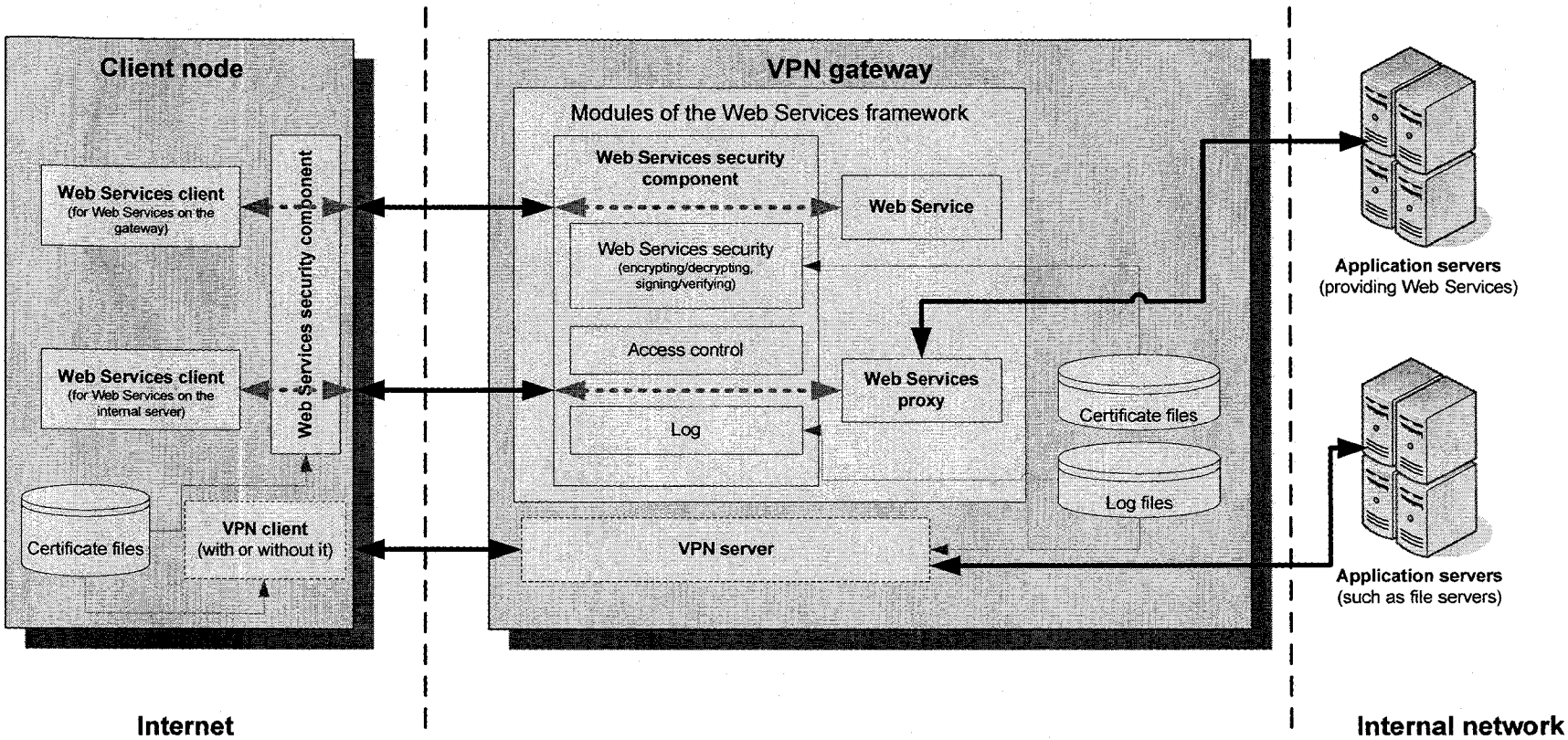

Figure 8 Message flow and relationship between different modules in the integrated security solution

4.2.1 Data flows in the VPN framework

In the VPN framework, VPN clients send connection requests to the VPN server to establish a secure communication channel. Once the VPN clients are authenticated and authorized, the access to the internal application servers are granted.

Data exchanged between the VPN clients and the VPN servers are sent over the secured VPN channel. Data exchanged between the VPN server and the internal application servers are not secured because this communication channel is assumed secure.

4.2.2 Message flows in the Web Services framework

There are two round-trip message flows of Web Services between the Web Services clients and the gateway. One contains Web Services requests and responses to and from the Web Service on the gateway. The other contains Web Services requests and responses to and from the Web Services proxy on the gateway.

In order to request the Web Service on the gateway, the Web Services client first build a SOAP message of Web Services request. This message then is encrypted and signed by the Web Services security component, using the key file on the client node. Next, the message is sent to the gateway. Upon receiving this message, the Web Services security component verifies the digital signature, decrypts the message and applies other security measures such as logging/auditing and access control. Then the message is sent to Web Services. The Web Service processes the request and then returns a response message. The response message is then processed subsequently by the Web Services security component on the gateway and the Web Services security component on the client. At last, the Web Services client gets the response message.

In order to request Web Services in the internal network, the Web Services client first sends a request to the Web Services proxy. Then the Web Services proxy sends request to the internal application server. Message exchanged between the Web Services proxy and the internal application servers is not processed by the Web Services security component. The request message and response message between the Web Services client and the

Web Services proxy are processed in the same way that the message between the Web Services client and the Web Service on the gateway is processed.

4.3 Design of the Web Services framework

This framework includes Web Services clients, Web Services on the gateway and on the internal application server, the Web Services proxy and the Web Services security component. The majority of the work of this thesis project is about the Web Services security component.

4.3.1 Web Services clients

Processing the signing/verifying and encrypting/decrypting SOAP message on the Web Services client is done by using WSS4J. Architecture of the Web Services client is depicted in Figure 9.

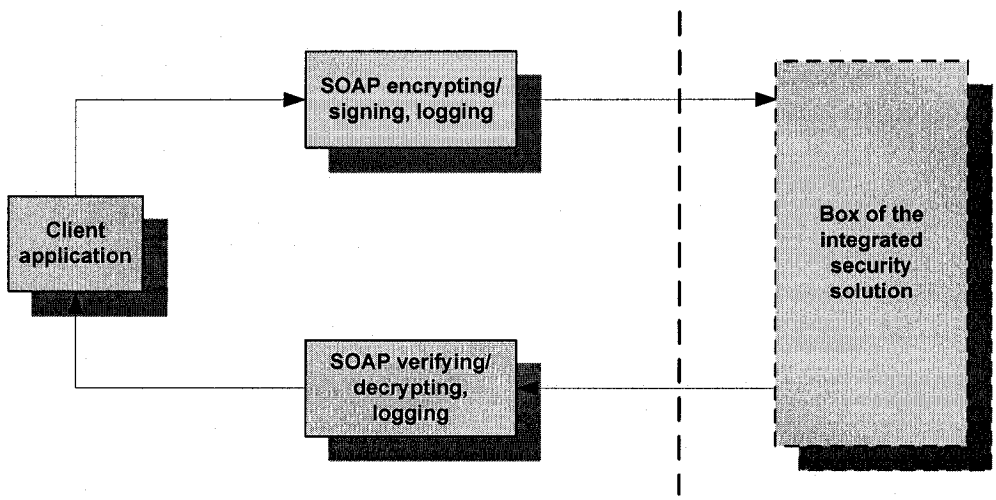

Figure 9 Architecture of the Web Services client

The Web Services client first calls Web Services on the gateway. Web Services are called through a SOAP message. This message will go through two steps of processing in a SOAP message handler defined by the WSDD file. In the first step, the operation is logged. In the second step, the SOAP message will be encrypted and signed with the sender's key file. Then the SOAP message is sent to Web Services on the gateway.

Upon receiving the reply from the gateway, also two steps of processing are applied to the SOAP message before it reaches the end point. The first one is in charge of logging. The second one is for verifying the digital signature and decrypting the SOAP message. Then the "plain" SOAP message with the reply from the gateway is sent to the end point, where we see the results of calling Web Service on the gateway.

There are two similar Web Services clients in the Web Services framework, one for requesting Web Services on the gateway, the other for requesting Web Services on the internal application servers.

4.3.2 Web Services on the gateway

We only build simple Web Services on the gateway, as the complexity of Web Services would not have any impact on the performance of the Web Services security component. We build simple Web Services on the gateway that return some information about BUL to Web Services clients.

Web Services on the gateway are protected by the Web Services security component. The requests to and the responses from Web Services on the gateway must go through the Web Services security component, where the security measures are applied.

4.3.3 Web Services on the internal application server

Web Services on the internal application servers are similar to Web Services on the gateway. The difference is that no security measures are applied on the internal Web Services application servers as the internal network is considered a "safe" place. Another reason is that it is efficient to apply security measures only on the gateway.

Web Services on the internal application servers are also Web Services who need to be protected by the Web Services security component too. In a real world, these Web Services are usually not visible by Web Services clients outside the gateway and should be mapped to the gateway. There can be many application servers that provide Web Services to the outside world. However, there are still other application servers that are not aware of Web Services.

In this thesis, Web Services on the internal application server are simulated by Web Services on the gateway for simplicity. This will not affect the complexity of the problem, as it is the same to connect the Web Services proxy and the Web Service via HTTP protocol, no matter where these two reside.

4.3.4 Web Services security component

The Web Services security component refers to the function units on the gateway that apply security measures to the SOAP messages. Processing of the Web Services security component is denoted by the solid boxes in Figure 10.

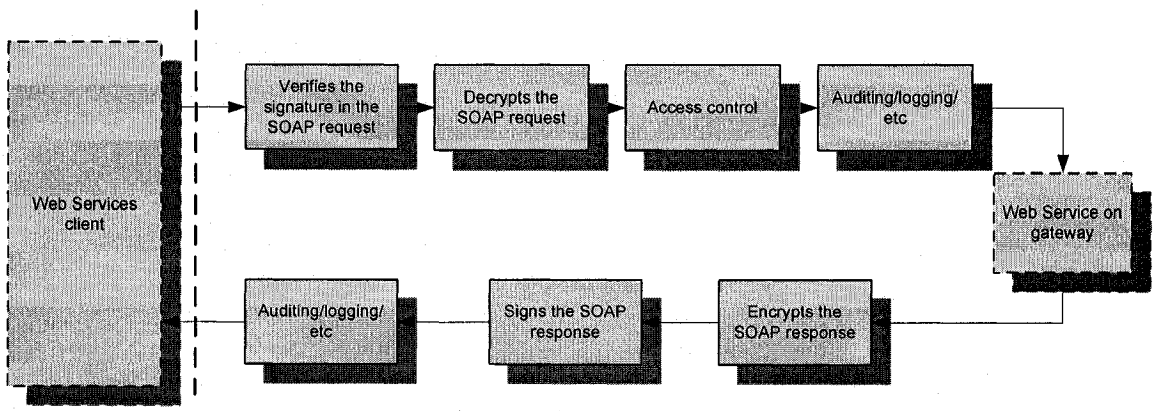

Figure 10 Processing of the Web Services security component

Upon receiving the SOAP message from the Web Services client, the Web Services security component will first verify the signature contained in the SOAP message. If the signature is not valid, the request is rejected. The identity contained in the SOAP message is used for access control and auditing/logging if the signature verification is successful.

Then the SOAP message is decrypted. The encrypted data, usually the method of the target Web Services to invoke, is extracted.

After access control and auditing/logging, the request is processed by the target Web Services and the results are returned to the Web Services client. Before sending, the SOAP message is encrypted and signed and the transaction is audited and logged, as shown in Figure 10.

Design issues of the Web Services security component are detailed in Chapter 5.

4.3.5 Web Services proxy

The Web Services proxy is also a Web Service. It receives the Web Services requests from the Web Services client and requests another Web Services on the internal application servers for the Web Services client. The Web Services proxy is depicted in Figure 11.

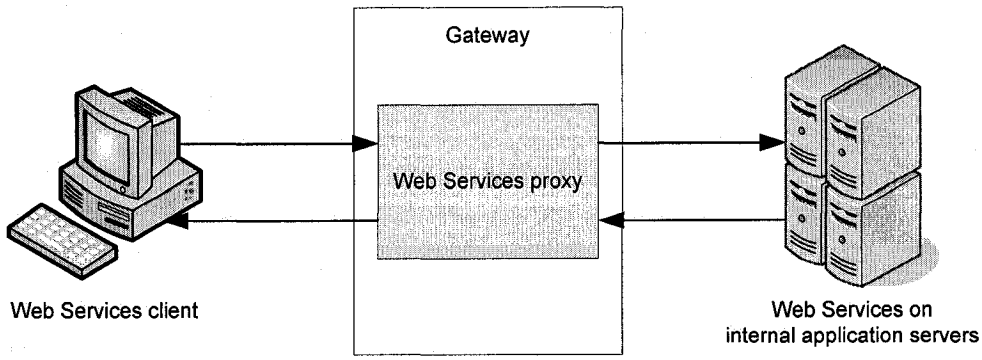

Figure 11 Function of the Web Services proxy

The Web Services proxy in our thesis is a prototype of a Web Services gateway operating in a real deployment environment. The Web Services gateway performs as follows.

If the application servers in the internal network are aware of Web Services, the Web Services gateway maps Web Services requests from outside the gateway to requests to the internal application servers. The Web Services proxy is needed, as usually the structure of the internal network cannot be seen outside of the gateway and the Web Services gateway (or a group of Web Services gateways) is the only point that is exposed on the Internet.

For legacy applications on the internal application servers, the Web Services gateway acts as an interpreter between Web Services clients outside the gateway and the legacy applications. When requests for Web Services come to the gateway, the Web Services gateway analyzes the requests and converts them to requests to legacy applications, e.g., SQL queries to database servers. When the legacy applications return the results, the Web Services gateway converts the result to replies to the Web Services requests and then sends the replies back to the Web Services clients.

In this thesis, the Web Services proxy is only in charge of mapping Web Services requests from the Web Services client to requests to Web Services on the gateway, as Web Services on the internal application server are simulated by Web Services on the gateway. Moreover, we do not consider support for legacy applications at the present time. All these are left for future research and development.

4.4 Chapter summary

Functions of the components of the integrated security gateway were described in this chapter.

In the Web Services framework, the Web Services clients request information from Web Services on the gateway or Web Services in the internal network. Web Services proxy acts as an agent between Web Services clients and the Web Services in the internal network. The Web Services on the gateway and the Web Services proxy are protected by the Web Services security component.

Chapter 5 Design of the Web Services Security Component

The design of the Web Services security component, also the major focus of this thesis, is described in this chapter.

5.1 Overview

The Web Services security component is in charge of enforcing security measures on XML Web Services traffic. XML traffic that does not meet certain security criteria (e.g., invalid digital signature, invalid identity) is blocked before it comes to Web Services on the gateway and on the Web Services proxy for Web Services in the internal application servers.

The Web Services security component integrated with the VPN server, will utilize the same set of ECC keys.

5.2 Web Services security with ECC

This section is dedicated to describing how Web Services are secured with Elliptic Curve Cryptography. Bell University Lab has elliptic curve Certificate Authority (CA) software and utilizes the ECC keys on the VPN gateway. The VPN gateway utilizes ECC keys mainly for its higher efficiency over RSA systems. As we will integrate the Web Services security component with the VPN server, it uses the same set of ECC keys that are used by the VPN servers.

5.2.1 Elliptic curves systems

Elliptic curves cryptography systems are based on the algebraic structure of elliptic curves over finite fields. The use of elliptic curves in cryptography was suggested by Neal Koblitz [35] and Victor S. Miller [36] in 1985.

ECC systems are believed to be more efficient than RSA in terms of memory usage and CPU time. Key sizes of 163 bits as in ECC ensure the security of a RSA key of 1024 Bit, which is used as a standard today. With greater key sizes, ECC systems are even better. A comparison of ECC and RSA key sizes in depicted in Table 1 (extracted from [43]).

ECC Key Size (Bits)	RSA Key Size (Bits)	Key Size Ratio	AES Key Size (Bits)
163	1024	1:6	
256	3072	1:12	128
384	7680	1:20	192
512	15360	1:30	256

Table 1 Comparison of RSA and ECC key sizes

Popular packages with ECC support include OpenSSL [10] and Crypto++ [38]. OpenSSL is an open source library and utility written with C. Crypto++ is a free library written with C++.

5.2.2 ECIES and ECDSA

1) ECIES

Elliptic Curve Integrated Encryption Scheme (ECIES) public key encryption scheme is also called the Elliptic Curve Augmented Encryption Scheme or simply Elliptic Curve Encryption Scheme. ECIES is specified in ANSI X9.63 [37] and the IEEE Standard 1363a [44] Draft. ECIES combines elliptic curve asymmetric encryption and the AES symmetric encryption algorithm with the SHA-1 hash algorithm to provide an easy to use encryption scheme with message authentication support. In this scheme, the encrypting party uses its own private key and the other party's public key to calculate a shared secret key that can be used as the key for symmetric encryption.

As a prerequisite to user A sending user B a message encrypted with ECIES, user A should have knowledge of the following:

- EC domain parameters agreed upon by both parties, that is (p,a,b,G,n,h) in the case of the prime field or $(m,f(x),a,b,G,n,h)$ in the case of binary field
- User B's public key
- A Message Authentication Code (MAC) scheme to be used, for example, HMAC-SHA-1-160 (HMAC [46] using SHA-1 [45] the key size of 160 bits)
- A Key Derivation Function (KDF), for example, ANSI-X9.63-KDF with SHA-1 option. KDF is used to generate keying data from some shared information between the participating parties
- Other shared information

1) Algorithm of ECIES encryption

Below is the algorithm of ECIES encryption, as stated in ANSI X9.63 and used in this thesis as the encryption scheme for Web Services security.

Input:

1. A bit string *EncData* to be encrypted
2. Recipient's public key Q
3. (Optional) Two bit strings of data, *SharedData*1 and *SharedData*2 shared by user A and user B

Output:

The bit string $QE \| MaskedEncData \| MacTag$ as the encryption of *EncData* (where || stands for concatenation)

Encryption procedure:

1. Generate a key pair (d_e, Q_e) with private key d_e and public key Q_e

2. Convert the public key Q_e to a bit string QE

3. Derive a shared secret field element z from d_e and Q

4. Convert z to a bit string Z

5. Use the KDF with the established hash function to generate keying data $KeyData$ from Z and optional $SharedData1$. Parse $KeyData$ as an encryption key $EncKey$ and a MAC key $MacKey$, i.e., parse $KeyData$ as:

$$KeyData = EncKey \,||\, MacKey$$

6. Compute $MaskedEncData = EncData \oplus EncKey$ where $\oplus$ stands for bitwise exclusive-or

7. Compute the tag $MacTag$ on the bit string:

$$MacData = MaskedEncData \,||\, SharedData2$$

(inclusion of $SharedData2$ is optional)

2) Algorithm of ECIES decryption

Below is the algorithm of ECIES decryption, as stated in ANSI X9.63 and used in this thesis.

Input:

1. A bit string $QE \,||\, MaskedEncData \,||\, MacTag$ supposed to be the ECIES encryption of a bit string

2. An EC private key d owned by the recipient

3. (Optional) Two bit strings of data, $SharedData1$ and $SharedData2$ shared by the sender and the recipient

Output:

EncData as the ECIES decryption of $QE \| MaskedEncData \| MacTag$

Decryption procedure:

1. Validate the public key Q_e extracted from the input of the decryption procedure. Stop if the validation fails.
2. Derive a shared secret field element z from d and Q_e.
3. Convert z to a bit string Z.
4. Use the KDF function with the established hash function to generate keying data *KeyData* from Z and optional *SharedData*1 . Parse *KeyData* as an encryption key *EncKey* and a MAC key *MacKey*, i.e. parse *KeyData* as:

$$KeyData = EncKey \| MacKey$$

5. Compute $EncData = MaskedEncData \oplus EncKey$.
6. Verify that *MacTag* is the tag on $MaskedEncData \| SharedData2$ under the key *MacKey* (inclusion of *SharedData*2 is optional). The decryption procedure fails if the tag checking outputs "invalid."

2) ECDSA

ECDSA is the elliptic curve analogue of the Digital Signature Algorithm (DSA). It was first proposed in 1992 by Scott Vanstone [11]. It is standardized in the ANSI X9.62 [47], NIST FIPS 186-2 [48], and IEEE 1363-2000 [49].

1) Algorithm of ECDSA signature generation

Input:

1. The message, M, to be signed, which is represented by a bit string
2. A set of elliptic curve domain parameters

3. An elliptic curve private key, d, associated with the elliptic curve domain parameters

Output:

Two integers r and s (the digital signature), where $1 \le r \le n-1$, $1 \le s \le n-1$

Procedure:

1. Message digesting. Compute the hash value $e = H(M)$ using the hash function SHA-1. e is represented as an integer with a length of 160 bits.
2. Select a statistically unique and unpredictable integer k in the interval $[1, n-1]$.
3. Compute the elliptic curve point $(x_1, y_1) = kG$.
4. Convert the field element x_1 to an integer $\overline{x_1}$. Set $r = \overline{x_1} \bmod n$. If $r = 0$, go to Step 2.
5. Compute $s = k^{-1}(e + dr) \mod n$. If $s = 0$, go to Step 2.

2) Algorithm of ECDSA signature verification

Input:

1. The received message, M', represented as a bit string
2. The received signature for M', represented as the two integers, r' and s'
3. A set of elliptic curve domain parameters
4. A public key, Q, associated with the elliptic curve domain parameters, used to verify the signature

Output:

Verification result, "valid" or "invalid"

Procedure:

1. Message digesting. Compute the hash value $e' = H(M')$ using the hash function SHA-1. e' is represented as an integer with a length of 160 bits.

2. If r' is not an integer in the interval $[1, n-1]$, the signature is invalid.

3. If s' is not an integer in the interval $[1, n-1]$, the signature is invalid.

4. Compute $c = (s')^{-1} \mod n$.

5. Compute $u_1 = e'c \mod n$ and $u_2 = r'c \mod n$.

6. Compute the elliptic curve point $(x_1, y_1) = u_1 G + u_2 Q$. If $u_1 G + u_2 Q$ is at infinity, the signature is invalid.

7. Convert the field element x_1 to an integer $\overline{x_1}$.

8. Compute $v = \overline{x_1} \mod n$.

9. If $r' = v$, the signature is valid. Otherwise, it is not valid.

5.2.3 XML security

XML security is the application of data privacy and authentication to XML structures. XML Security with ECC is done by applying ECC encryption algorithm ECIES to XML Encryption and applying ECC signature algorithm ECDSA to XML Signature. In this thesis, the ECC encryption and signature algorithm are both on the binary fields.

1) XML Encryption

The XML Encryption Syntax and Processing recommendation [41] defines a process for encrypting digital data and how the resulting encrypted data should be represented in XML. XML Encryption supports the encryption of the entire XML document or any parts of it.

An encrypted element is contained in the structure *CipherData*. *CipherData* may optionally include *EncryptionMethod*, *KeyInfo* and *EncryptionProperties*. A sample XML document with XML Encryption is depicted in Listing 2.

```
<apache:RootElement xmlns:apache="http://www.apache.org/ns/#encsample1">
  <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Type="http://www.w3.org/2001/04/xmlenc#Content">
    <xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"
      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
      <xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        <xenc:EncryptionMethod
          Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes"
          xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
        <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          <xenc:CipherValue xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
            lzbuDJCSIbQ9zDImaW08Krup2B18nPTfTAMY2RJDKeM=
          </xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedKey>
    </ds:KeyInfo>
    <xenc:CipherData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
      <xenc:CipherValue xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        ...
      </xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</apache:RootElement>
```

Listing 2 Sample XML document with elements of XML encryption

Below are the steps to encrypting an XML document.

1. Select the encryption algorithm and parameters.
2. Obtain the key.
3. Encrypt the data.
4. Build the *EncryptedType* structure to store the encrypted data.
5. Replace the unencrypted element with *EncryptedType* in the XML document.

To decrypt an XML document, the following procedure is run.

1. Obtain the decryption key. The decryption key may be the private key of the receiver of the XML document or an encrypted secret key.
2. Decrypt the data contained in *CipherData* using the algorithm specified in the XML document and the parameters specified in the XML document or the application.

3. Process of the decrypted data. This usually includes converting the decrypted data from the UTF-8 encoded form to its original form.

The Apache XML Security library implements XML Encryption and is used in this thesis project as a means of encrypting/decrypting SOAP messages.

2) XML Signature

The W3C XML-Signature Syntax and Processing recommendation [40] defines how digital data is signed and how the resulting signature should be represented in XML. With XML Signature, all or selected parts of the XML document can be signed.

XML Signature is composed of two elements, *SignedInfo* and *SignatureValue* . *SignedInfo* contains information about the canonicalization method, signature algorithm and one or more references to the data being signed. *SignatureValue* is Base64 [26] encoded value of the digital signature. A sample XML document with XML Signature is shown in Listing 3.

```
<apache:RootElement Location="SITE" bul:Location="SITE5026">Bell University Lab
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod
        Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315">
      </CanonicalizationMethod>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1">
      </SignatureMethod>
      <Reference URI="http://www.w3.org/TR/xml-stylesheet">
        <DigestMethod
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
        </DigestMethod>
        <DigestValue>60NvZvtdTB+7UnlLp/H24p7h4bs=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>
      E50B74wXb+Dqyosx5go1xeOYnGA4VXU5Gto4ezDJ6GgiD8X+a5vjeg==
    </SignatureValue>
    <KeyInfo>
      <KeyValue>
        <DSAKeyValue>
          <P>
            ...
          </P>
          <Q>
            ...
          </Q>
          <G>
            ...
          </G>
          <Y>
            ...
          </Y>
        </DSAKeyValue>
      </KeyValue>
    </KeyInfo>
  </Signature>
</apache:RootElement>
```

Listing 3 Sample XML document with elements of XML Signature

The procedure to sign an XML document is as follows.

1. Apply the transform or transforms to the data object to be signed. Transforms are applied in the order they are specified.
2. Calculate the message digest of the output of the transforms.
3. Create one or more reference elements that include the URI of the data object, the transforms used, the digest algorithm and the digest value.
4. Create the *SignedInfo* element that includes the *SignatureMethod* , the *CanonicalizationMethod* and the references previously generated.
5. Canonicalize *SignedInfo* .

6. Use the algorithms specified by *SignatureMethod* to create the signature.

7. Create the *Signature* element that contains the *SignedInfo* , the *SignatureValue* and the *KeyInfo* .

To verify the signature in an XML document, the procedure below is followed.

1. Canonicalize the *SignedInfo* element according to the *CanonicalizationMethod* specified in *SignedInfo* .

2. For each reference element, obtain the data object referenced.

3. Process each data object according to the specified transforms.

4. Digest the results according to the digest algorithm specified for the referenced element. Compare the results with the value stored in the corresponding reference element. The verification procedure continues only if these two values are the same.

5. Obtain the necessary keying information.

6. Apply the signature method using the previously obtained key to confirm the *SignatureValue* over the canonicalized *SignedInfo* element.

The Apache XML Security library implements XML Signature and is used in this thesis project as the means of creating/verifying digital signature in SOAP messages.

5.2.4 SOAP message security

SOAP message security is implemented by applying security measures to messages in SOAP format. It is built on top of XML security. SOAP message security with ECC is done by applying ECC encryption algorithm ECIES and ECC signature algorithm ECDSA to SOAP messages through XML security.

1) SOAP encryption

SOAP encryption is stated in the OASIS specification about SOAP message security [32]. The process of encrypting a SOAP message is described as follows.

1. Create a new SOAP envelope.
2. Create a <wsse:Security> header.
3. When an <xenc:EncryptedKey> is used, create a <xenc:EncryptedKey> sub-element of the <wsse:Security> element.
4. Locate data items to be encrypted, i.e., XML elements, element contents within the target SOAP envelope.
5. Encrypt the data items using XML encryption algorithms.

Decryption can likewise be done.

2) SOAP signature

Method for SOAP signature is also stated in the OASIS specification about SOAP message security [32]. The process of adding a digital signature to a SOAP message is described below.

The procedure to add a digital signature to a SOAP message is as follows.

1. Create a new SOAP envelope.
2. Create a <wsse:Security> header.
3. Add the reference to the data to be signed.
4. Add the signature of data.

Verification of the digital signature in a SOAP message can be done likewise.

5.2.5 Web Services security

As security for other applications over insecure communication channels, identification and authentication, authorization, confidentiality, integrity and non-repudiation must be ensured for Web Services. Web Services security with ECC in this thesis is done by SOAP message encryption and signature with ECC algorithms.

1) Identification and authentication

One communicating party should verify that other parties are who they claim to be before granting any right of access; this is called identification and authentication. The verified identity is used for authorization and for audit.

Identification and authentication should be done with the identity contained in the certificate and digital signature. Upon receiving the certificate of the sender of a SOAP message, the receiver will use the identity contained in the certificate and check the digital signature of the SOAP message using the public key of the sender stored at the receiver side. If the signature is valid, the sender is authenticated.

2) Authorization

Authorization is the process to grant permission for principals to access resources after identification and authentication. It provides the basis of access control. The purpose of access control is to ensure that only authorized principals may access the resources that they are allowed to access.

Authorization does not use SOAP message encryption and signature with ECC algorithms directly, but it depends on the results of the verification of the digital signature in the SOAP messages.

3) Confidentiality

Confidentiality is defined as preventing unauthorized entities from reading or learning about certain resources, usually the data stored or being exchanged. Confidentiality with ECC algorithms is implemented by applying of ECC algorithm ECIES in SOAP messages.

4) Integrity

During transmission, data may be altered by other parties. Integrity is about preventing data from being modified without being detected by the legitimate recipient of the data.

Data integrity in Web Services security with ECC can be ensured by means of ECC signature algorithm ECDSA. Upon receiving a SOAP message with a digital signature,

the receiver verifies the digital signature in the SOAP message against the received message. If the digital signature is valid, then the receiver can tell that the message has not been altered by unauthorized parties.

5) Non-repudiation

Non-repudiation ensures that the sender of a message cannot deny sending the message.

Non-repudiation in Web Services security with ECC is provided by digital signature too. With the sender's signature on some SOAP messages, the receiver can make sure that the message is sent by the claimed sender by checking the digital signature.

5.3 Software suite

5.3.1 .NET framework and Java

Though the specifications and standards of Web Services and Web Services security are owing to efforts of many institutes, only two kinds of environments are commonly used to implement Web Services and Web Services security. One is .NET environment and the other one is Java environment.

.NET is Microsoft's implementation of a development platform for Internet applications using a suite of technologies, tools, products and protocols. The .NET platform provides a variety of implementations of Web Services and Web Services security technologies. It includes many servers and technologies, as shown below.

- .NET enterprise servers such as the Internet Security and Acceleration Server and the SQL Server.
- Windows servers such as Windows Server 2003.
- .NET framework, the run-time that supports the development and management of XML Web Services, including the Visual Studio .NET development environment.
- Other implementations for Web Services such as XML, SOAP, UDDI and WSDL.

The Visual Studio .NET and the .NET framework allows users to build Web Services with ASP .NET, Visual Basic .NET, Visual C# .NET and so on. With the .NET framework, we can build needed Web Services and ensure Web Services security. The Microsoft WS-I Basic Security Profile 1.0 Reference Implementation, for example, was developed with the .NET framework and Microsoft Web Services Enhancements.

However, the serious drawback of using the .NET framework to develop Web Services and ensure Web Services security is that it does not allow port among different platforms. Another problem is that we have to pay for almost everything, not only the tools we use to develop Web Services and ensure Web Services security, but also the deployment environment.

With Java, we can deploy it on different platforms with only a little effort. Moreover, there is open source or free supporting packages for developing Web Services with Java, e.g., Eclipse as a Java integrated development environment. The Apache Web Services project is a good open source development toolkit for Web Services and Web Services security. We used the software packages such as Axis, WSS4J and XML Security to build the Web Services security component in this thesis.

5.3.2 Java software suite for the Web Services security component

The Web Services component is built with several Java software packages. Apache Axis is used as the SOAP engine that is in charge of handling SOAP messages. Web Services are Servlets implemented in Axis running on Tomcat, which is a Servlet container. WSS4J implements Web Services security handlers. It calls the Apache XML Security library in turn for XML security APIs. The jBorZoi library provides ECC encryption and signature algorithms for XML security. The relationship between these software packages is depicted in Figure 12. Other supporting packages are omitted from this figure for clarity.

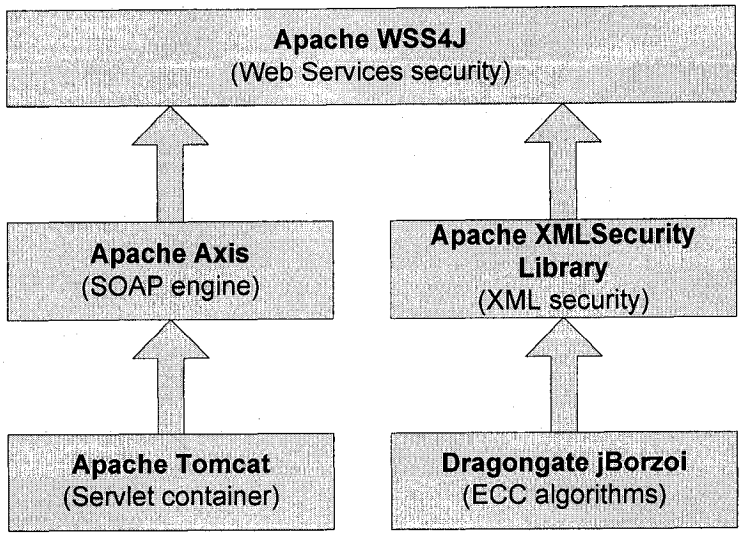

Figure 12 Relationship between software packages

As shown in Figure 12, the package suite needed for building the Web Services security component includes the following packages.

- Apache Tomcat as the servlet container.
- Apache Axis as the SOAP engine. Classes are in `org.apache.axis`.
- Apache WSS4J for SOAP message security. It calls classes and classes in the Apache XML Security package. Classes are in `org.apache.ws`.
- The Apache XML Security package for XML security. Classes are in `org.apache.xml.security`.
- Dragongate jBorzoi for ECIES algorithm and ECDSA algorithm for BUL ECC keys. Classes are in `com.dragongate_technologies.borZoi`.
- Other supporting packages such as Apache Log4J (supporting logging at runtime without modifying the application binary), Apache Jakarta Commons HTTP Client (for implementing Web Services client) and Apache Xerces Java Parser (as XML API for Java)

5.4 *Message handler chain in Apache Axis*

In Apache Axis, messages are processed by a series of processing units called handlers, as shown in Figure 13. Processing of security measures such as SOAP encryption, SOAP signature, access control and auditing/logging can be done in handlers. By adopting Apache WSS4J, processing of both SOAP encryption and SOAP signature is done in one handler implemented by `bul.wsgw.handler.WSDoAllReceiver` (adapted from `org.apache.ws.axis.security.WSDoAllReceiver`) and `bul.wsgw.handler.WSDoAllSender` (adapted from `org.apache.ws.axis.security.WSDoAllSender`).

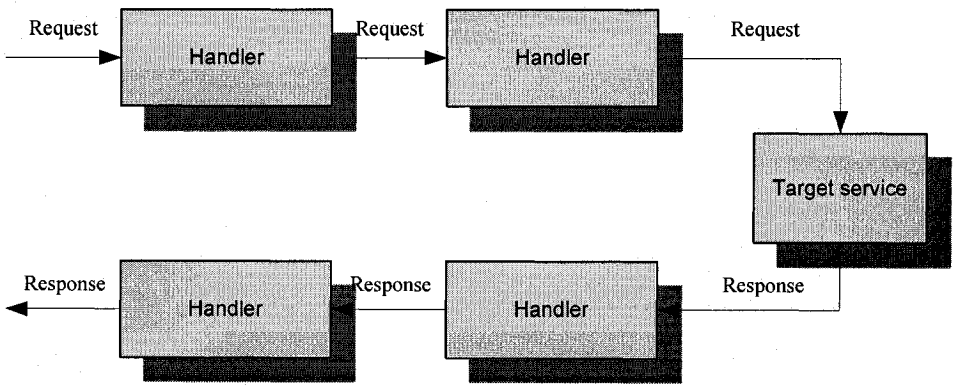

Figure 13 SOAP message handlers in Apache Axis

The target service is called a pivot handler, which processes requests and gives responses. Other handlers process only requests or responses.

5.5 *SOAP message handling in WSS4J*

The processing of signing/verifying and encrypting/decrypting SOAP messages is done by using WSS4J to adopt the BUL ECC keys. The processing of SOAP messages in WSS4J is depicted in Figure 14.

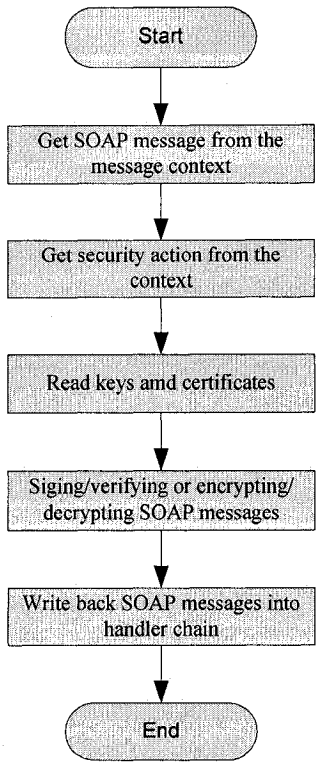

Figure 14 Processing of SOAP message security

First we need to get the SOAP message from Axis message context. Then we read the security action from the context, signature or encryption. Next we read keys or certificates from key file. According to the security action indicated in the context, we may encrypt the SOAP message, decrypt the SOAP message, sign the SOAP message or verify the digital signature of the SOAP message. At last, we write the processed SOAP message back into the message context.

5.6 Chapter summary

Functions of the Web Services security component were detailed in this chapter.

Web Services security is basically SOAP message security. For Web Services security, we needed to solve the problem of ECC encryption and signature algorithms. Then we could apply these algorithms to XML security. Next we could apply XML security to SOAP message security.

In the Java software suite to implement Web Services security, SOAP message are processed in handlers in Apache Axis. WSS4J, which is in charge of SOAP message security, is called in Axis's handlers.

Chapter 6 Implementation of the Integrated Security Solution

This chapter is dedicated to the implementation of the integrated security solution for securing both traditional network-based applications and Web Services. Our focus in this chapter is on the Web Services framework, especially the Web Services security component.

There are six components implemented in this Web Services framework. They are the Web Services client for Web Services on the gateway, the Web Services client for Web Services on the internal application servers behind the gateway, Web Services on the gateway, Web Services on the internal application servers, the Web Services proxy and, most important, the Web Services security component. As their names suggest, the two Web Services clients and two Web Services work in pairs. The Web Services proxy on the gateway is also a Web Service; it calls Web Services on the internal application servers upon receiving requests from Web Services clients and returns the results to the Web Services clients. In this way, it works as a "proxy." The Web Services security component is the major focus of this thesis project.

In this chapter, a description of the implementation platform is followed by the implementation details of every module.

6.1 Implementation platform

The Web Services clients, Web Services, the Web Services proxy and the Web Services security component are implemented with Java. So the operating system does not matter. The Web Services client is implemented and run on Windows XP. The components on the gateway side are implemented and run on a Linux system with kernel version 2.4.23. The version of the other packages is described in Table 2.

Role	Package with version number
Java Development Kit	JDK 1.5.0
Java Runtime Environment	JRE 1.5.0
Servlet container	Apache Tomcat 5.5.0
SOAP engine	Apache Axis 1.3.0 (implements SOAP version 1.2)
Web Services Security	Apache WSS4J 1.1.0
XML Security	Apache XML Security 1.3.0
ECC Algorithms	Dragongate jBorzoi 0.90

Table 2 Packages for building the Web Services framework

6.2 Implementation of the Web Services framework

Implementation of the Web Services framework includes implementation on the client side and implementation on the gateway side. There are two Web Services clients implemented on the client side, one for the Web Service on the gateway and the other for the Web Service on the internal application server. On the gateway side, there are the Web Service on the gateway, the Web Service on the internal application server, the Web Services proxy and the Web Services security component.

6.2.1 Web Services clients

The Web Services clients call Web Services on the gateway or on the internal application servers. They are implemented on Apache Axis. As there are also security measures (SOAP encryption and signature) applied on SOAP messages on the client side, there is a security handler similar to the Web Services security component on the gateway.

A part of the code of the Web Services client for Web Services on the gateway is shown in Listing 4.

```
public static void main(String[] args)
throws ServiceException, RemoteException
{
    if (args.length == 0)
    {
        System.out.println("Usage:\njava ClientGateway [fullname | location]");
        return;
    }
    ServiceGatewayService locator = new ServiceGatewayServiceLocator();
    ServiceGateway service = locator.getServiceGateway();
    String returnedInfo = service.showBULInfo(args[0]);
    System.out.println("Information about BUL,  " + args[0] + ": " + returnedInfo);
}
```

Listing 4 Definition of the Web Services client

6.2.2 Web Service on the gateway

The Web Service on the gateway returns results to a Web Services client. It is implemented on Apache Axis as a Servlet. There is not much business logic implemented in this Web Service as it is not related to the majority of the work of this thesis project. It simply returns some information about BUL. A part of the code of the Web Service on the gateway is shown in Listing 5.

```
public String showBULInfo (String typeInfo)
throws Exception
{
        if (typeInfo.equals("fullname") )
        {
                return "Bell University Labs";
        }
        if (typeInfo.equals("location"))
        {
                return "SITE 5026, University of Ottawa";
        }
        return "don't know what information you want";
}
```

Listing 5 Definition of the Web Service on the gateway

The Web Services security component is applied to Web Services by claiming the security component to be the handler of Web Services. The deployment file is shown in Listing 6.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://
xml.apache.org/axis/wsdd/providers/java">
 <service name="service-gateway" provider="java:RPC" style="document" use="literal">
  <requestFlow>
   <handler type="java:bul.wsgw.handler.WSDoAllReceiver">
    <parameter name="passwordCallbackClass"
      value="PWCallback"/>
    <parameter name="user" value="ServerTest"/>
    <parameter name="action" value="Signature Encrypt"/>
    <parameter name="decryptionPropFile" value="crypto.properties.server" />
    <parameter name="signaturePropFile" value="crypto.properties.server" />
  </handler>
  </requestFlow>
  <responseFlow>
   <handler type="java:bul.wsgw.handler.WSDoAllSender">
    <parameter name="passwordCallbackClass"
      value="PWCallback"/>
    <parameter name="user" value="ServerTest"/>
    <parameter name="action" value="Signature Encrypt"/>
    <parameter name="decryptionPropFile" value="crypto.properties.server" />
    <parameter name="signaturePropFile" value="crypto.properties.server" />
  </handler>
  </responseFlow>
  <parameter name="className" value="bul.wsgw.framework.gateway.ServiceGateway"/>
  <parameter name="allowedMethods" value="showBULInfo"/>
  <parameter name="scope" value="application"/>
 </service>
</deployment>
```

Listing 6 Deployment of the Web Services on the gateway

6.2.3 Web Service on the internal application server

The Web Service on the internal application server works with the Web Services proxy. Upon receiving Web Services requests from the Web Services client, the Web Services proxy (which is also a Web Service) sends requests to the Web Service on the internal server. When the results from the internal application server are ready, the Web Services proxy sends back the results to the Web Services client.

The Web Service on the internal application server is simulated by a Web Service on the gateway for simplicity. This does not affect the complexity of the problem, as the Web Services proxy and the Web Service on the internal application server are connected by HTTP protocol, and there is no difference no matter where the Web Service resides.

The implementation of the simulating Web Service is similar to the implementation of the Web Service on the gateway and is omitted.

6.2.4 Web Services security component

The Web Services security component is implemented by a SOAP message handler and includes three steps of processing. When a SOAP message comes from a Web Services client, it is processed as follows.

The first step is in charge of SOAP message security, which is SOAP encryption/decryption and SOAP message signing and SOAP digital signature verifying.

The second step is in charge of access control. This is based on the security policies shared by the VPN server and the Web Services security component.

The transaction is logged in the third step.

When a SOAP message leaves for a Web Services client, there are only two steps of processing, as there should not be access control for an outgoing message.

The implementation of the Web Services security component is detailed step by step in Section 6.3.

6.2.5 Web Services proxy

The Web Services proxy is also a Web Service. Upon receiving Web Services requests, it calls another Web Service in turn and returns the results from the called Web Services to the Web Services client.

There is not much business logic implemented by this Web Services proxy.

6.3 Implementation of the Web Services security component

The implementation of the processing of the Web Services security component is described in detail here.

The basis of the SOAP message security is acquiring keys from PEM files of our lab. Next is the implementation of ECIES and ECDSA with these keys. Adoption of these algorithms in XML security is then implemented, followed by implementation of SOAP message security through XML security. At last, integration of the Web Services security component with the VPN server is described.

6.3.1 Reading BUL ECC keys

The basis of processing Web Services security is reading the keys and certificates. We utilize the BUL ECC keys for signing/verifying and encrypting/decrypting SOAP messages. The keys and certificates are stored in PEM files. The processing of reading BUL ECC private keys from PEM files is depicted in Figure 15.

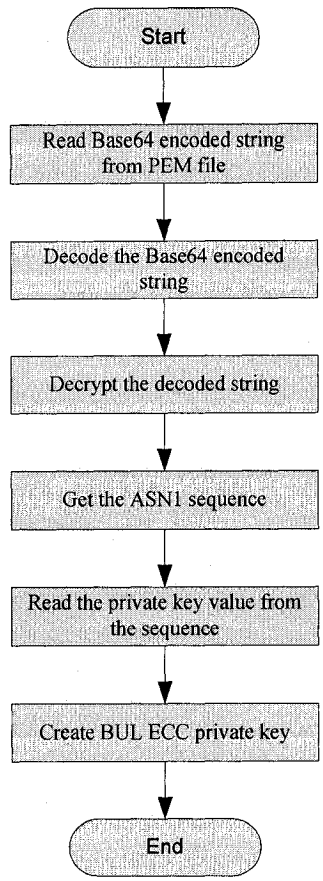

Figure 15 Reading BUL ECC private keys from PEM files

BUL ECC keys are stored in PEM files using X.509 [30] format. These PEM files are Base64 encoded data files. In these PEM files, private keys or certificates are in data sections of the PEM files. Data elements in each section, such as private key, common name, public key and EC domain parameters are stored in a data structure called the Abstract Syntax Notation One (ASN.1) sequence. An ASN.1 [51] sequence from the decrypted private key section in a BUL PEM file is shown in Listing 7. It is obtained with the OpenSSL utilities. The item shown as "OCTET STRING" is the private key.

```
  0:d=0  hl=2 l=  82 cons: SEQUENCE
  2:d=1  hl=2 l=   1 prim:  INTEGER           :01
  5:d=1  hl=2 l=  20 prim:  OCTET STRING      [HEX DUMP]:3FA7D042752E326FA30733869D0B7F87B24CF972
 27:d=1  hl=2 l=   7 cons:  cont [ 0 ]
 29:d=2  hl=2 l=   5 prim:   OBJECT           :sect163r1
 36:d=1  hl=2 l=  46 cons:  cont [ 1 ]
 38:d=2  hl=2 l=  44 prim:   BIT STRING
```

Listing 7 ASN.1 sequence in the section of "EC PRIVATE KEY"

The basic idea behind our function of reading private key or certificate is to locate the data elements in the data structure.

In order to read the private key from the PEM files, we first read the Base64 encoded string from the PEM file. Second, we decode the Base64 encoded string. Then we decrypt the decoded string before getting the ASN.1 sequence. Then we read the private key value from the sequence. From the private key value and the elliptic curve domain parameters of BUL ECC keys, we can get the ECC private keys.

The processing of BUL certificates is similar. As the BUL ECC certificates are in the standard X.509 format and the XML Security library can process the X.509 certificates directly, we do not have to extract data elements such as the common name, public key, ASN.1 OID and subject. The work for us to do is to extract the data in the certificate section in the PEM file and pass the data to a *CertificateFactory* provider. Then an X.509 class can be obtained from this *CertificateFactory* provider. Here we use the *CertificateFactory* provider from BouncyCastle [31].

6.3.2 ECIES and ECDSA on elliptic curves with EC domain parameter recommendation SECT163R1

In this thesis project, we used the BUL ECC key for this purpose. The BUL ECC keys use SECT163R1 as the recommendations for EC domain parameters. SECT163R1 recommends a set of parameters for ECC keys with key size of 163-bit.

To the best of our knowledge, there is no support for XML security with such EC domain parameters in known Java packages. We provide the support for such keys by using the Dragongate jBorzoi package to support elliptic arithmetics for SECT163R1.

6.3.3 Processing of XML encryption

Processing XML encryption is the basis of processing SOAP encryption in WSS4J. Processing of XML encryption is done by using the Apache XML Security library to add the ability of utilizing BUL ECC keys and certificates. The processing of XML encryption is depicted in Figure 16.

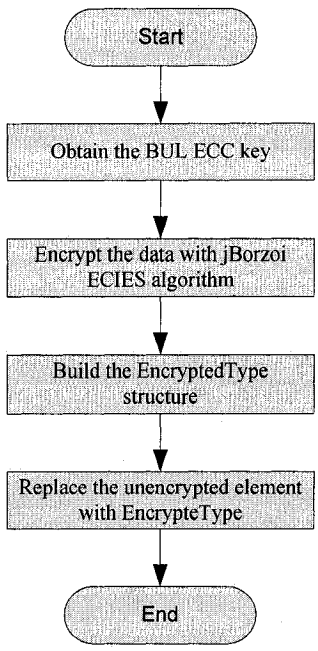

Figure 16 Processing of XML encryption

Encrypting XML starts with obtaining the encrypting key. With support for SECT163R1, we can then use the public key cryptographic algorithm ECIES supported in the jBorzoi package for use with XML encryption. We then convert the encrypted data into the ASN.1 sequence. At last we pack the ASN.1 sequence into the *CipherValue* element.

To decrypt an XML document, we first obtain the encrypted data from the encrypted element in the XML document before we convert the encrypted data into the ASN.1 sequence. We then convert the ASN.1 sequence into the jBorzoi ECIES class. At last we can decrypt the data using the proper decrypting key.

6.3.4 Processing of SOAP encryption

SOAP encryption with BUL ECC keys is done by adapting the WSS4J package, which in turn calls the XML Security library.

In order to encrypt a SOAP document, we need to first set the encoding namespace in the SOAP envelope. Then we get the data to be encrypted before we encrypt the data using ECIES. After data encryption we set up the `wsse:Security` header block.

The difference between the standard WSS4J processing of SOAP encryption and our scheme of SOAP encryption with BUL ECC keys is the creation of the symmetric encryption key is done by the jBorzoi ECIES class with the encrypting party's private key and the other party's public key in our scheme.

Decrypting a SOAP message is done as follows. We first get the *CipherValue* in the XML message. Then we decode the Base64 encoded data in *CipherValue* and get the ASN.1 sequence represented by the data. We then get the decrypting key (reception's private key). The jBorzoi ECIES class is converted from the ASN.1 sequence and then decrypted with the decrypting key.

6.3.5 Processing of XML signature

In this thesis project, processing of XML signature is done by using the Apache XML Security library to add the ability of utilizing BUL ECC keys and certificates. In order to utilize BUL ECC keys, the jBorzoi ECDSA algorithm is called. The processing of XML signature is depicted in Figure 17.

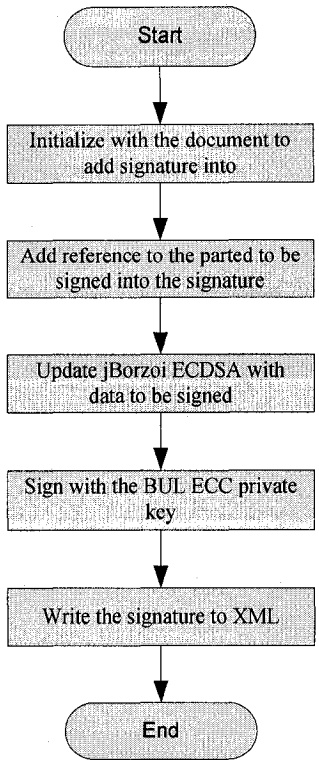

Figure 17 Processing of XML signature

The addition of a signature to the XML document begins by initializing the `XMLCipher` class with the document to be signed. Then we add a reference to the elements to be signed into the signature. After that, we update the jBorzoi ECDSA algorithm with the data to be signed. Then we sign with the ECDSA class. At last, we write the signature into the XML document.

When verifying an XML signature, we first get the signature data from the XML message and decode the Base64 encoded data. We then convert the data into an ASN.1 sequence. Next, we convert the ASN.1 sequence into a jBorzoi ECDSA class. Lastly, we can verify the signature with the signer's public key.

6.3.6 Processing of SOAP signature

SOAP signature using BUL ECC keys is processed by using the Apache WSS4J library.

To sign a SOAP message, we need to first set the encoding namespace in the SOAP envelope. Then the signature element is created using the XML signature algorithm that

supports BUL ECC keys. At last, we insert the signature element into the `wsse:Signature` header block.

6.3.7 Integration with the VPN server

In the Web Services security component integrated with the VPN server, processing of access control and auditing/logging are added to the handler.

For auditing/logging, signing/verifying and encrypting/decrypting, the requested Web service and result of access control are recorded.

For access control, Web Services requests with invalid digital signatures are rejected. More sophisticated rules can be implemented in the future.

6.4 Running results

Running the Web Services framework includes two pieces of work, one for calling Web Services on the gateway, the other for calling Web Services on the internal application servers behind the gateway through the Web Services proxy on the gateway. As the results for calling these two types of Web Services are basically the same, we show only the process of calling the Web Services on the gateway in this section.

6.4.1 Calling a Web Service on the gateway

1) Deploy the Web Services client

This Web Services client is to call the Web Service on the gateway. This is done by first locating the Web Service and then calling the method of the Web Service. As there are security measures applied on the SOAP messages on the Web Services client, a WSDD file is defined, as shown in Listing 8.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://
xml.apache.org/axis/wsdd/providers/java">
 <transport name="http" pivot="java:org.apache.axis.transport.http.HTTPSender"/>
  <globalConfiguration >
   <requestFlow >
    <handler type="java:bul.wsgw.handler.WSDoAllSender" >
     <parameter name="action" value="Signature Encrypt"/>
     <parameter name="passwordType" value="PasswordDigest"/>
    <parameter name="user" value="clienttest"/>
    <parameter name="passwordCallbackClass"
value="bul.wsgw.framework.client.PWCallback"/>
    <parameter name="encryptionPropFile" value="crypto.properties.client" />
    <parameter name="signaturePropFile" value="crypto.properties.client" />
    </handler>
   </requestFlow >
   <responseFlow >
    <handler type="java:bul.wsgw.handler.WSDoAllReceiver" >
     <parameter name="action" value="Signature Encrypt"/>
     <parameter name="passwordType" value="PasswordDigest"/>
    <parameter name="user" value="clienttest"/>
    <parameter name="passwordCallbackClass"
value="bul.wsgw.framework.client.PWCallback"/>
    <parameter name="encryptionPropFile" value="crypto.properties.client" />
    <parameter name="signaturePropFile" value="crypto.properties.client" />
    </handler>
   </responseFlow >
  </globalConfiguration >
</deployment>
```

Listing 8 WSDD file for the Web Services client

2) Deploy the Web Service

Before we can call the Web Service, we must deploy it on the server, which is implemented by Tomcat. As described in the previous sections, there is a handler in the request flow of this Web Service. This handler is in charge of SOAP decryption and SOAP signature verification, access control and logging the transaction. The WSDD file is shown in the Listing 6. The command used to deploy the Web Service is shown in Listing 9.

```
java org.apache.axis.client.AdminClient -lhttp://137.122.88.134:8080/axis/services/
AdminService bul\wsgw\framework\gateway\deploy2.wsdd
```

Listing 9 Deployment of the Web Service on the gateway

3) Bind the Web Service

The `org.apache.axis.wsdl.WSDL2Java` is a tool for creating classes for calling a Web Service from the WSDL file of the Web Service. In order to create the classes for our Web Services client, we run the `WSDL2Java` tool, as shown in Listing 10.

```
java org.apache.axis.wsdl.WSDL2Java -o . -Nhttp://137.122.88.134:8080/axis/services/
service-gateway bul.wsgw.framework.client http://137.122.88.134:8080/axis/services/
service-gateway?wsdl
```

Listing 10 Use WSDL2Java to generate the client service bindings

4) Call the Web Service

Now we can call Web Services by running the Web Services client. The command to call the Web Service on the gateway is shown in Listing 11.

```
java -Daxis.ClientConfigFile=client_deploy.wsdd bul.wsgw.framework.client.ClientGateway fullname
```

Listing 11 Call the Web Service on the gateway

5) SOAP message processed by the Web Services client

The part of the SOAP message that is signed and encrypted by the Web Services client is shown in Listing 12 and Listing 13. The X509 serial number of the client certificate is 41. These results show that the SOAP message is signed and encrypted with the correct key file.

```
<ds:SignatureValue xmlns="" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
                MDswCQYHKoZIzj0CATAuAhUCOP8jTLCoxX2xBfXG0C4dPh0VLPICFQFAvfs1gnM5BCX7NEaHhaqT
                J9F7xw==
</ds:SignatureValue>
...
<ds:X509SerialNumber xmlns="" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">41
</ds:X509SerialNumber>
```

Listing 12 SOAP message signed by the Web Services client

```
<ds:X509SerialNumber xmlns:ds="http://www.w3.org/2000/09/xmldsig#">41
</ds:X509SerialNumber>
...
<xenc:CipherData>
        <xenc:CipherValue>
                MIIBEDCB1TCBogYHKoZIzj0CATCB1gIBATAaAgIAowYJKoZIzj0BAgMDMAkCAQMCAQYCAQcwLgQV
                B7aILKrvqE+VVP+EKL2I4kbSeCriBBUHE2Etzdy0CquUa9opypH3OvlYr9kEKwQDaZeWl6tDiXeJ
                VmeJVn94enh2plQAQ17bQu+vspidUf7848gJiPQf+IMCFQP/////////////SKq2icKcpxAnmwIB
                AgMuAAQrBALe1CEditd+q8MCOEIxdYRPzKZ8pwf+B77Pnf02VNSmvcRDC45LDXkcqQQgPv3vdlV0
                BkuJg/RQaZPvAMwNBznKJul0GbodJzU5o0MEFKZATjnscoSPVSh1FglcsqoHUkgP
        </xenc:CipherValue>
```

Listing 13 SOAP message encrypted by the Web Services client

6) SOAP message processed by the Web Service on the gateway

The part of the SOAP message that is signed and encrypted by the Web Service on the gateway is shown in Listing 14 and Listing 15. The X509 serial number of the certificate is 2. These results show that the SOAP message is signed and encrypted with the correct key file.

```
<ds:SignatureValue xmlns="" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance">
                MDswCQYHKoZIzj0CATAuAhUA8DGjUBjfVOFq2iW33jehX0gJpjcCFQNUfNS1hXZQ0Jwl+K9tcB6g
                +F1tGQ==
</ds:SignatureValue>
...
<ds:X509SerialNumber xmlns="" xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">2
</ds:X509SerialNumber>
```

Listing 14 SOAP message signed by the Web Service on the gateway

```
<ds:X509SerialNumber
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#">2
</ds:X509SerialNumber>
...
<xenc:CipherData>
        <xenc:CipherValue>
                MIIBEDCB1TCBogYHKoZIzj0CATCB1gIBATAaAgIAowYJKoZIzj0BAgMDMAkCAQMCAQYCAQcwLgQV
                B7aILKrvqE+VVP+EKL2I4kbSeCriBBUHE2Etzdy0CquUa9opypH3OvlYr9kEKwQDaZeWl6tDiXeJ
                VmeJVn94enh2plQAQ17bQu+vspidUf7848gJiPQf+IMCFQP/////////////SKq2icKcpxAnmwIB
                AgMuAAQrBASe3USyzwDK+bKT1r/YpVfwFxE9nwCyjVb+12vgejclqH24B2Ok4qt9BgQgne45cTkL
                Bnq+LHhR6FI58O1+xYoFFR7WDhd0v+5uNcYEFELUaVCvPDOIGIj8dWB+eTLy/ASf
        </xenc:CipherValue>
</xenc:CipherData>
```

Listing 15 SOAP message signed by the Web Service on the gateway

7) Get the results

Then we can get the results from Web Services on the gateway. The results are shown in Listing 16 as expected.

```
C:\Tomcat5.0\webapps\axis\WEB-INF\classes\bul\wsgw\framework\client>java -
Daxis.ClientConfigFile=client_deploy.wsdd bul.wsgw.framework.client.ClientGateway fullname

Information about BUL,  fullname: Bell Unversity Labs
```

Listing 16 Returned results of the Web Service on the gateway

6.5 Performance

The Web Services security component is basically an implementation of the elliptic curve encryption algorithm ECIES and the elliptic curve signature algorithm ECDSA. Then it utilizes these two algorithms to process SOAP messages, which are XML documents by nature.

The complexity of this Web Services security component then depends on the complexity of algorithms ECIES and ECDSA. While adapting the Apache XML Security library and the Apache WSS4J library, we do not add any processing that will add complexity.

As the processing of SOAP messages (encryption and signature) is similar on the client side and on the gateway side, we did the experiment on one personal computer to measure the speed. On a PC with an AMD 1.2GHz CPU and Windows XP, it takes about 30 seconds for a Web Services client to get the results, which is run on the same PC. The estimated time for the Web Services security component to process a request is about half of the total time, 15 seconds in our experiment.

The processing time seems too long to us. Even on a much faster computer, e.g., a personal computer with Intel 2.6GHz, it may still take several seconds to process a request. With the much better performance of ECC over RSA in mind, the same Web Services security component that is based on RSA would be even much slower.

For a Web Services security component in a real deployment environment, the cryptographic engine has to be hardened to get much better performance. That is why the cryptographic engines of all the commercial Web Services security gateways are implemented by hardware.

6.6 Chapter summary

The implementation of the Web Services security component was detailed in this chapter.

Apache Axis, Apache WSS4J, Apache XML Security library and Dragongate jBorzoi library were chosen for Web Services security. First we needed to modify jBorzoi library to support ECC encryption and signature algorithms with BUL ECC keys. Then we applied these two algorithms to XML security library for XML encryption and XML signature. Next, we called XML security functions in WSS4J for SOAP message security.

The experiment result showed that the SOAP messages were correctly signed/verified and encrypted/decrypted on both client side and server side.

Chapter 7 Conclusion and Future Work

The thesis project is summarized and directions of future research are given in this chapter.

7.1 Thesis summary

This thesis was to propose an integrated security solution that included a VPN framework and a Web Services framework. The major focus was the research on and implementation of a Web Services security component integrated with the BUL's VPN gateway. This Web Services security component utilized ECC keys over binary fields from BUL. Our implementation of the Web Services security component supported ECIES as the algorithm for SOAP message encryption and decryption and ECDSA as the algorithm for SOAP message signing and signature verification.

We first introduced research related to this thesis. Web Services are based on XML, which is an open standard for application-to-application communication interface. The components in a Web Services architecture, SOAP, WSDL and UDDI are all based on XML. SOAP provides a means of communication between Web Services. Web Services security was implemented by SOAP message security, including SOAP encryption and SOAP signature. SOAP encryption and SOAP signature are in turn based on XML encryption and XML signature. The Apache Web Services suite is a good software package for implementation of Web Services security.

What needed to be addressed for Web Services security were five points: identification and authentication, authorization, confidentiality and non-repudiation. These five points were provided with SOAP encryption and SOAP signature.

Our implementation of the Web Services security component utilized the BUL ECC keys for a better integration with the existing BUL'S VPN gateway, which utilized BUL ECC

keys. These ECC keys conform to SECT163R1 recommendation for EC domain parameters.

The basis of the Web Services security component was reading ECC keys and certificates from the key files. For reading private keys from a PEM file, we first read all the data in the private key section and then decrypted the data before we got the private key value from the data structure.

Then we added a set of EC domain parameters in the Dragongate jBorzoi package to support the BUL ECC keys.

What we did next was XML encryption and XML signature with BUL ECC keys. The work included understanding the XML security standards and adding support for algorithms ECIES and ECDSA.

For SOAP message security, SOAP encryption and SOAP signature supporting the BUL ECC keys were implemented. This included adding security headers in the SOAP message and adding encryption add/or signature elements in the SOAP message.

The Web Services security component was then integrated into the BUL'S VPN gateway.

The Web Services framework was developed using the Apache Web Services suite and other Java packages. This Web Services framework included a Web Service on the gateway and the corresponding Web Services client, a Web Service on the internal application server and the corresponding Web Services client, a Web Services proxy mapping Web Services on the internal application servers to Web Services on the gateway and the Web Services security component. The running of the framework showed that the results are as expected.

7.2 Future work

The theories and technologies for the Web Services security have been studied in this thesis. Based on the Apache Web Services suites and other Java packages, a Web Services security component integrated with the BUL'S VPN gateway has been developed. Integration of the Web Services security component with the VPN server has been proven to be feasible and efficient. This integrated security solution can be

improved to make a network device that has the functions of a VPN gateway, a Web Services security gateway and a Web Services gateway.

A simple Web Services security provider has been implemented, which processes PEM files to get BUL ECC keys or certificates. A better choice would be full-featured key store that supports BUL ECC keys.

The Web Services security component has been implemented by WSS4J, which in turn calls functions in the Apache XML Security library. The way that these libraries utilize BUL ECC keys is by the jBorzoi packages. A better choice would be integrating BUL ECC keys into the Java cryptography engine so the BUL ECC keys do not need special processing in WSS4J or XML Security libraries.

The security policies on the Web Services security component and the VPN server are quite simple at this time. There have to be comprehensive security policies if this integrated gateway is to be applied in an enterprise. Moreover, these policies have to be consistent with the existing policies and may have to combine the existing security policies. So the integrated gateway will have to add support for LDAP and SAML.

When the VPN traffic and the Web Services traffic is high, modules such as cryptography that impose a heavy burden on the CPU have to be implemented using hardware.

References

[1] Extensible Markup Language (XML) 1.0, the World Wide Web Consortium
http://www.w3.org/TR/xml/

[2] Rolf Oppliger, "Security Technologies for the World Wide Web", Artech House, 2003.

[3] Man Young Rhee, "Internet Security: Cryptographic Principles, Algorithms, and Protocols", John Wiley, 2003.

[4] Jalal Feghhi, Jalil Feghhi and Peter Williams, "Digital Certificates: Applied Internet Security", Addison-Wesley, 1999.

[5] Frank P. Coyle, "XML, Web Services, and the Data Revolution", Addison-Wesley, 2002

[6] William R. Cheswick, Steven M. Bellovin and Aviel D. Rubin, "Firewalls and Internet Security, Repelling the Wily Hacker", Addison Wesley, 2003.

[7] Product whitepapers from Datapower Technology, Inc.
http://www.datapower.com

[8] Forum XWall™ Datasheet, Forum Systems
http://forumsystems.com/papers/XWall_Data_Sheet_Spring%20_2004.pdf

[9] James Yonan, "The User-Space VPN and OpenVPN"
http://openvpn.net/papers/BLUG-talk/BLUG-talk.ppt

[10] OpenSSL
http://www.openssl.org/

[11] S. Vanstone, "Responses to NIST's proposal". Communications of the ACM, 35:50-52, July 1992.

[12] Juniper NetScreen
http://www.juniper.net/products/ssl/

[13] S. Kent and R. Atkinson, "RFC2401 - Security Architecture for the Internet Protocol", Internet Engineering Task Force, 1998
http://www.ietf.org/rfc/rfc2401.txt?number=2401

[14] IPSec vs. SSL VPNs for Secure Remote Access, Aventail White Paper
http://www.aventail.com/downloads/pdfs/IPSECvsSSL_WP.pdf

[15] Bret Hartman, Donald J. Flinn, Konstantin Beznosov and Shirley Kawamoto, "Mastering Web Services Security", Wiley Publishing, Inc. 2003

[16] W3C SOAP Version 1.2
http://www.w3.org/TR/soap12-part0/

[17] UDDI Executive Overview: Enabling Service-Oriented Architecture, UDDI white paper
http://uddi.org/pubs/uddi-exec-wp.pdf

[18] WSDL Version 2.0, W3C working draft
http://www.w3.org/TR/wsdl20-primer

[19] W3C XML Encryption Workgroup
http://www.w3.org/Encryption/2001/

[20] W3C XML Digital Signature Workgroup
http://www.w3.org/Signature/

[21] W3C XML Key Management Working Group
http://www.w3.org/2001/XKMS/

[22] IEEE P1363: Standard Specifications for Public-Key Cryptography, Institute of Electrical and Electronics Engineers, 2000.

[23] SEC 2: Recommended Elliptic Curve Domain Parameters, Standards for Efficient Cryptography Group, working draft, September 2000

[24] Darrel Hankerson, Alfred Menezes and Scott Vanstone, "Guide to Elliptic Curve Cryptography", Springer-Verlag New York, Inc. 2004.

[25] Dragongate jBorzoi Java Elliptic Curve Cryptography Library
http://dragongate-technologies.com/products.html#jBorZoi

[26] Nathaniel Borenstein and Ned Freed, "RFC1521 - MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", Internet Engineering Task Force, September 1993.

[27] Data Encryption Standard, Federal Information Processing Standards Publication 46-1, US National Institute of Standards and Technology 1988.

[28] Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, US National Institute of Standards and Technology, May 2004.

[29] Advanced Encryption Standard (AES). Federal Information Processing Standards Publication 197, US National Institute of Standards and Technology, 2001.

[30] Russell Housley, Warwick Ford, Tim Polk and David Solo, "RFC3280 - Internet X.509 Public Key Infrastructure: Certificate and CRL Profile", Internet Engineering Task Force, April 2002

[31] JCE Provider from Legion of Bouncy Castle

http://www.bouncycastle.org/

[32] OASIS Web Services Security: SOAP Message Security 1.0 Standard 200401, Organization for the Advancement of Structured Information Standards, March 2004.

[33] Tim Dierks and Christopher Allen, "RFC2246 - The Transport Layer Security Protocol version 1.0", Internet Engineering Task Force, 1999

http//:www.ietf.org/rfc/rfc2246.txt

[34] Taher ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, Advances in Cryptology", Proceedings of CRYPTO '84, pp10–18, Santa Barbara, California, USA, August 19-22, 1984

[35] N. Koblitz, "Elliptic Curve Cryptosystems", in Mathematics of Computation 48, 1987, pp. 203–209

[36] V. Miller, "Use of Elliptic Curves in Cryptography", CRYPTO 85, 1985.

[37] ANSI X9.63, Public Key Cryptography For The Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography, American National Standards Institute, January 1999

[38] Crypto++ library of Cryptographic Schemes

http://www.eskimo.com/~weidai/cryptlib.html

[39] SEC 1: Elliptic Curve Cryptography, Standards for Efficient Cryptography Group, working draft, September 2000

[40] XML-Signature Syntax and Processing, W3C Recommendation, 12 February 2002.

[41] XML Encryption Syntax and Processing, W3C Recommendation 10 December 2002

[42] Whitfield Diffie and Martin E. Hellman, "New Directions in Cryptography, IEEE Transactions on Information Theory", vol. IT-22, Nov. 1976, pp: 644-654.

[43] Jerry Krasner, "Using Elliptic Curve Cryptography (ECC) for Enhanced Embedded Security", November 2004

http://www.embeddedforecast.com/EMF-ECC-FINAL1204.pdf

[44] IEEE Std 1363a-2004, IEEE Standard Specifications for Public-Key Cryptography - Amendment 1: Additional Techniques, Institute of Electrical and Electronics Engineers, September 2, 2004

[45] FIPS PUB 180-1: Secure Hash Standard, US National Institute of Standards and Technology, April 1995.

[46] FIPS PUB 198: The Keyed-Hash Message Authentication Code (HMAC), US National Institute of Standards and Technology, March 2002.

[47] ANSI X9.62, Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), American National Standards Institute, September 20, 1998

[48] FIPS 186-2, Digital Signature Standard (DSS), American National Standards Institute, January 27, 2000.

[49] IEEE Std 1363-2000, IEEE Standard Specifications for Public-Key Cryptography, Institute of Electrical and Electronics Engineers, January 30, 2000

[50] John Franks, et al, "RFC2617 - HTTP Authentication: Basic and Digest Access Authentication", Internet Engineering Task Force, June 1999

[51] ITU-T X.680, Information Technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation, International Telecommunication Union Telecommunication standardization Sector, July 2002

[52] Dale Husemöller, "Elliptic Curves", Second Edition, Springer-Verlag New York, Inc., 2004

[53] Vivek Chopra, Amit Bakore, Jon Eaves, Ben Galbraith, Sing Li and Chanoch Wiggers, "Professional Apache Tomcat 5", Wiley Publishing, Inc., 2004

[54] David Chappell and Tyler Jewell, "Java Web Services", O'Reilly, March 2002.

[55] James McGovern, Sameer Tyagi, Michael Stevens and Sunil Matthew, "Java Web Services Architecture", Morgan Kaufmann Publishers, 2003.

[56] Jonathan B. Knudsen, "Java Cryptography", O'Reilly, May 1998

[57] Ramesh Nagappan, Robert Skoczylas and Rima Patel Sriganesh, "Developing Java Web Services, Architecting and Developing Secure Web Services Using Java", Wiley Publishing Inc., 2003

[58] Brett McLaughlin, "Java and XML", Second Edition, Wiley Publishing Inc., 2001

[59] RSA-200 is factored!, RSA Laboratories

http://www.rsasecurity.com/rsalabs/node.asp?id=2879

[60] Darui Xu, "A Java Implementation of the Simple Object Access Protocol", MSc thesis, Florida State University, October 2002.