DEVELOPMENT OF A WEB SERVICES SECURITY ARCHITECTURE BASED ON .NET FRAMEWORK

A Thesis Submitted to
The Graduate School of Engineering and Sciences of
Izmir Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Software

by
Recep BACI

October 2008
İZMİR

We approve the thesis of **Recep BACI**

_____
Assist. Prof. Dr. Tuğkan TUĞLULAR
Supervisor

_____
Assoc. Prof. Dr. Ahmet KOLTUKSUZ
Committee Member

_____
Assist. Prof. Dr. Gökhan DALKILIÇ
Committee Member

07 October 2008

_____
Prof. Dr. Sıtkı AYTAÇ
Head of the Computer Engineering of Department

_____
Prof. Dr. Hasan BÖKE
Dean of the Graduate School of Engineering and Sciences

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Assist. Prof. Dr. Tuğkan TUĞLULAR, for his guidance, patience and encouragement. He was the one who supported me when I was in trouble with critical decisions. His valuable support and confidence have been the driving force of this thesis work.

I would also like to thank Msc. Statistician Kıvanç YÜKSEL who cooperated with me in statistical studies.

Finally, I should thank to my family who always supported me throughout my education in my graduate study.

ABSTRACT

DEVELOPMENT OF A WEB SERVICES SECURITY ARCHITECTURE BASED ON .NET FRAMEWORK

Service Oriented Architecture (SOA) is an architectural style which allows interaction of diverse applications regardless of their platform, implementation languages and locations by utilizing generic and reliable services that can be used as application building block. SOA includes methodologies and strategies to follow in order to develop sophisticated applications and information systems. SOA is different from the traditional architectures as it has its own unique architectural characteristics and regulations, which needs to be analyzed and clarified so as to apply the information that should be included in the architectural model of SOA correctly to service based application development. The newest technology for SOA is web service technology which gains more and more importance as a technology to develop distributed service-oriented applications. Web services are an emergent paradigm for implementing business collaborations over the web. Each service has an interface that is accessible through standard protocols and that describes the interaction capabilities of the service.

This master's thesis primarily examines the web services concept of the .NET platform having the emphasis on secure communication. A case study demonstrates securing the communication between a web service and its clients through RIJNDAEL, 3DES and RSA algorithms implemented on code based structure which uses the identity token, provided from identity web service, to validate the identity of the client and the status token provided from status web service in order to validate the status of the client. A number of tests are performed using different cryptographic algorithms and network settings for the communication in order to obtain operational values of these algorithms.

ÖZET

.NET ÇERÇEVESİ TEMELLİ BİR WEB SERVİS GÜVENLİĞİ MİMARİSİ GELİŞTİRİLMESİ

Servis yönelimli mimari, platform, uygulama dilleri ve konumları ne olursa olsun uygulama temel ilke olarak kullanılabilen kapsamlı ve güvenli servislerden faydalanılarak farklı uygulamaların etkileşimine izin veren mimari bir stildir. Servis yönelimli mimari karmaşık uygulamalar ve bilişim sistemleri geliştirmek için yöntem ve stratejiler içerir. Servis yönelimli mimari geleneksel mimariden farklıdır çünkü doğru olarak servis bazlı uygulama geliştirmeye servis yönelimli mimarinin mimariye ilişkin modele dahil olması gereken bilgiyi uygulamak için analiz etmeye ve açıklamaya ihtiyaç duyulan kendine özgü eşsiz mimariye ilişkin karakteristiklere ve kurallara sahiptir. Servis yönelimli mimari için en yeni teknoloji dağıtık servis yönelimli uygulamaları geliştirmek için bir teknoloji olarak daha fazla önem kazanan web servis teknolojisidir. Web servisleri web üzerinde ticari ortaklıklar uygulamak için geliştirilen bir paradigmadır. Her servis standart protokollerce erişilebilinen ve servisin etkileşim kabiliyetlerini tanımlayan bir ara yüze sahiptir.

Bu tez öncelikle güvenli iletişimde önemle üzerinde durulan. NET platformu kullanılarak geliştirilen web servisleri kavramını incelemektedir. Yapılan örnek olay incelemesi, istemcinin statüsünün geçerliliğini denetlemek için, web servisince sağlanılan kimlik belirteci kullanılmış olup, kod bazlı yapılarca uygulanan RIJNDAEL, 3DES ve RSA algoritmaları kullanılarak, web servisleri ve onların istemcileri arasında güvenli iletişim sağlamayı ispatlamada kullanılmıştır. Bu algoritmaların işlevselliğini elde etmek için iletişim boyunca farklı kriptografik algoritmalar ve ağ ortamları kullanılarak bu algoritmaların operasyon değerlerinin belirlenmesi için testler yapılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

CHAPTER 1

INTRODUCTION

There is an increasing demand for technologies that support the connecting or sharing of resources and data in a very flexible and standardized manner. Because technologies and implementations vary across companies and even within divisions or departments, unified business processes could not be smoothly supported by technology. Integration is achievable only between units that are already aware of each other and that use the same static applications.

Web services technology, which enables disparate systems to interoperate at a high level with ease, lacked a common framework for security. Although some attempts have been made to solve this problem, most of these attempts bring a solution using variations of technologies already in use. These techniques, which are actually proven in the field, do not fit with the interoperable and loosely coupled nature of web services technology. The new emerging Web Service Security (WS-Security) standard, along with other extensions of the WS-* group of technologies aims this lack of a common security framework (Atkinson, et al. 2002).

The Web Services Security standard is formed by independent organizations backed by big vendors of the industry such as IBM, Microsoft, RSA, and Verisign. Although, some alignment in vision has been achieved, in practice there is an on going debate. This debate, although a good thing for advancement of technology, results in a slow penetration of technology. Proven products leveraging these technologies are still missing.

This thesis aims to comprehensive and extensible suggested web services security architecture and provide and provide an implementation, understanding and efficiency of this new technology. The design and architecture of the case study examined in this thesis, is implemented using a mix of several web services security specifications. Although there are alternative specifications still competing to become de facto standards, we believe the selection of specifications had recently gained momentum and success in becoming a de facto standard is in the near future.

CHAPTER 2

SERVICE ORIENTED ARCHITECTURE

2.1 Service Oriented Architecture Overview

Service Oriented Architecture (SOA) is an architectural style which utilizes methods and technologies that provides for enterprises to dynamically connect and communicate software applications between different business partners and platforms by offering generic and reliable services that can be used as application building blocks. In this way it is possible to develop richer and more advanced applications and information systems (Sun Microsystems Inc. 2006).

Software development turns out to be more challenging as the needs and desires grows to have complex infrastructures capable of solving real-world problems. Similarly, technological improvements through many tendencies and alternatives grounds to build compound architectures for developing software systems. The architecture of software explores the software system infrastructure by describing its components and high level interactions between each of them. These components are abstract modules built as a “unit” with other components. The high level interactions between components are called “connectors”. The configuration of components and connectors describes the way a system is structured and behaves (McGovern, et al. 2003), shown in Figure 2.1.

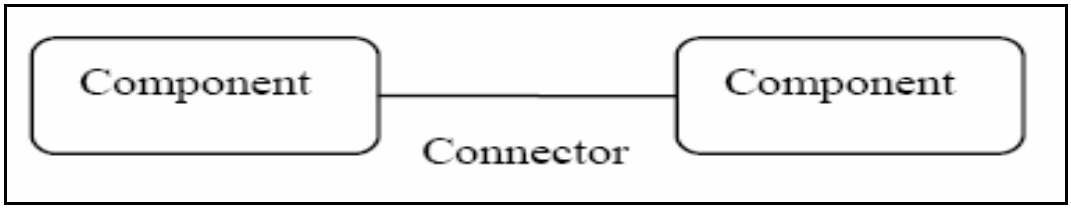

Figure 2.1. Abstract Definition of Software Architecture
(Source: McGovern, et al. 2003)

The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationship among them (Bass, et al. 1997). To simplify the complexity of the architecture, conventionally, the system is built with modules, which involves functions, objects, components and services.

Service Oriented Architecture (SOA) is a particular type of software architecture which has distinguished features and characteristics. The concept of SOA emerged in the early 1980s (Magedanz, et al. 2007) and become a significant architectural style especially after invention of web services. Before examining the architecture in detail, it is important to evaluate the existing software development concepts and related technologies to discover the revolution of SOA so as to not to develop SOA from scratch.

The concept of services is familiar to anyone who shops online at an e-commerce web site. Once you place your order, you have to supply your credit card information, which is typically authorized and charged by an outside service vendor. Once the order has been committed, the e-commerce company coordinates with a shipping service vendor to deliver your purchase. E-commerce applications provide a perfect illustration of the need for an SOA. If the credit card billing component is offline or unresponsive, you do not want the sales order process to fail. Instead, you want the order to be collected and the billing operation to proceed at a later time. Figure 2.2 provides a conceptual workflow for an e-commerce business that uses multiple services to process orders (Hasan 2006).

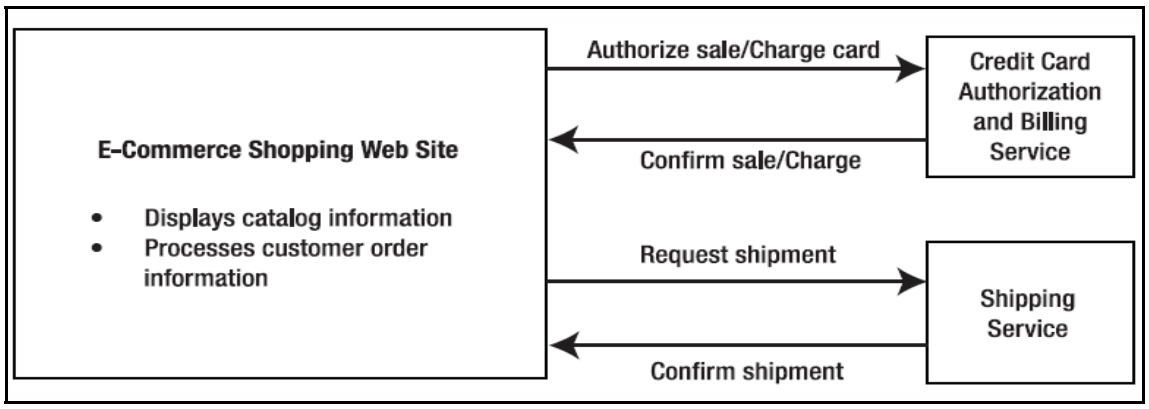

Figure 2.2. Service-based workflow for an e-commerce business
(Source: Hasan 2006)

Figure 2.3 shows a conceptual SOA that summarizes the three main entities in a typical SOA solution:

- Service providers
- Service consumers
- Service directories

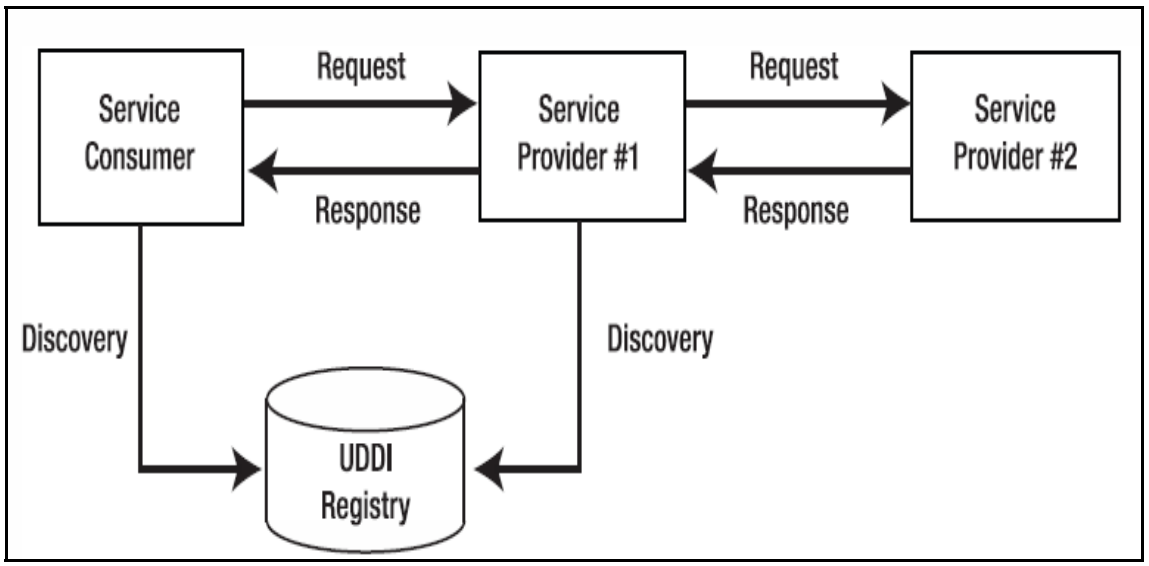

Figure 2.3. Conceptual SOA solution
(Source: Hasan 2006)

2.1.1 SOA Entities and Characteristics

Service Oriented Architecture is an architectural style that defines an interaction model between three main functional units, in which the consumer of the service interacts with the service provider to find out a service that matches its requirements through searching registry. A meta-model describing this interaction is shown in Figure 2.4 below.

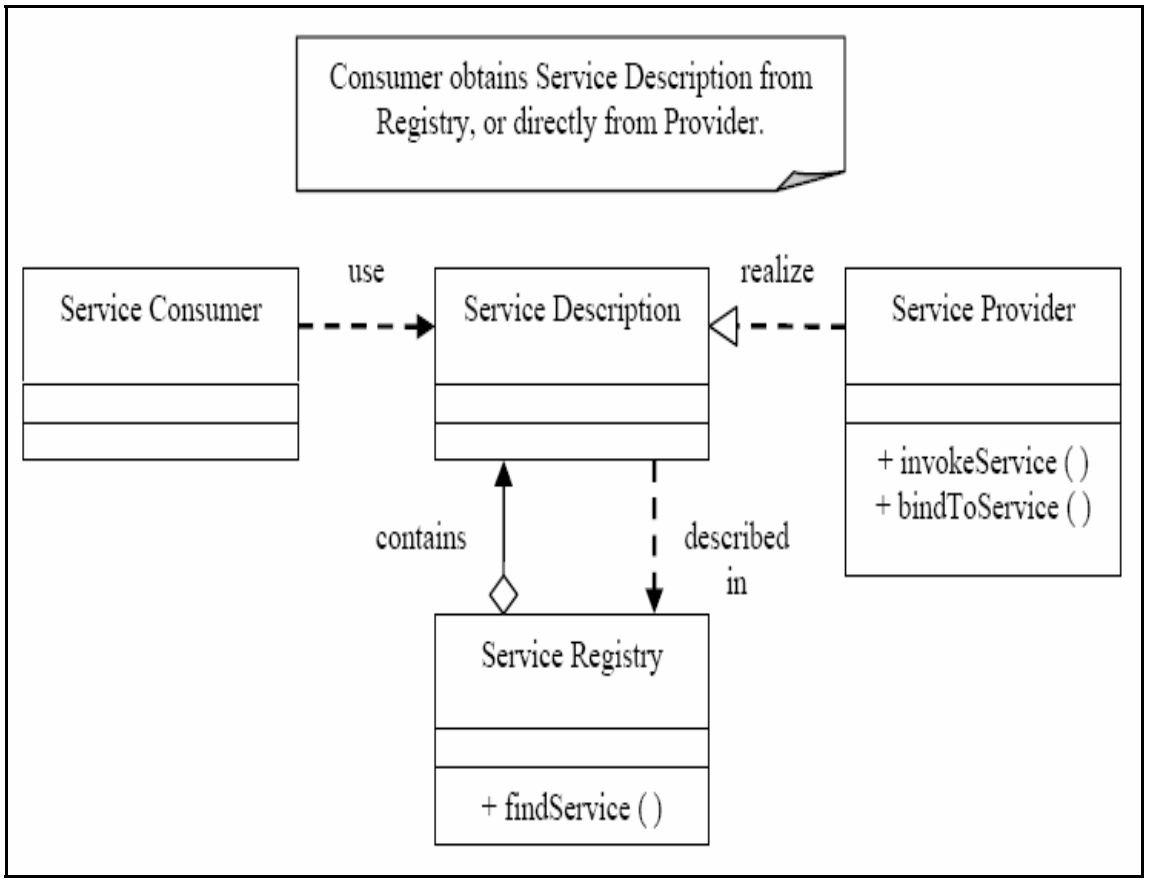

Figure 2.4. Service Oriented Architecture Conceptual Model
(Source: McGovern, et al. 2003)

SOA contains 6 entities in its conceptual model, described as follows (McGovern, et al. 2003):

- Service Consumer
- Service Provider
- Service Registry
- Service Contract
- Service Proxy
- Service Lease

Service oriented architecture reflects specific principles and characteristics that need to be applied when building service-oriented application infrastructures (Sun Microsystems 2001), which are described as follows:

- Services are discoverable and dynamically bound
- Services are self contained and modular
- Services are interoperable
- Services are loosely coupled
- Services have a network-addressable interface
- Services have coarse-grained interfaces
- Services are location transparent
- Services can be composed into new applications
- SOA supports self-healing

2.1.2 Service Oriented Development

Services are the evolution of components in which multiple component interfaces form into a single interface to perform a specific function. A service is an abstract resource with the capability of performing a task (Booth and Haas 2004). Services have the potential reflection of business functions as well as technical task definitions.

Services are designed and developed to support the following characteristics (Sehring 2006):

- Each service defines a specific business function and can match to real-life activities
- A service may have various procedures and operations
- Services interact with other services and system components in a loosely coupled, message-oriented environment to accomplish business goals
- Services has clearly defined interfaces and can be used by many different other services and applications
- Services do not need to be in a distributed environment

Figure 2.5 illustrates service based development in the context of components and objects.

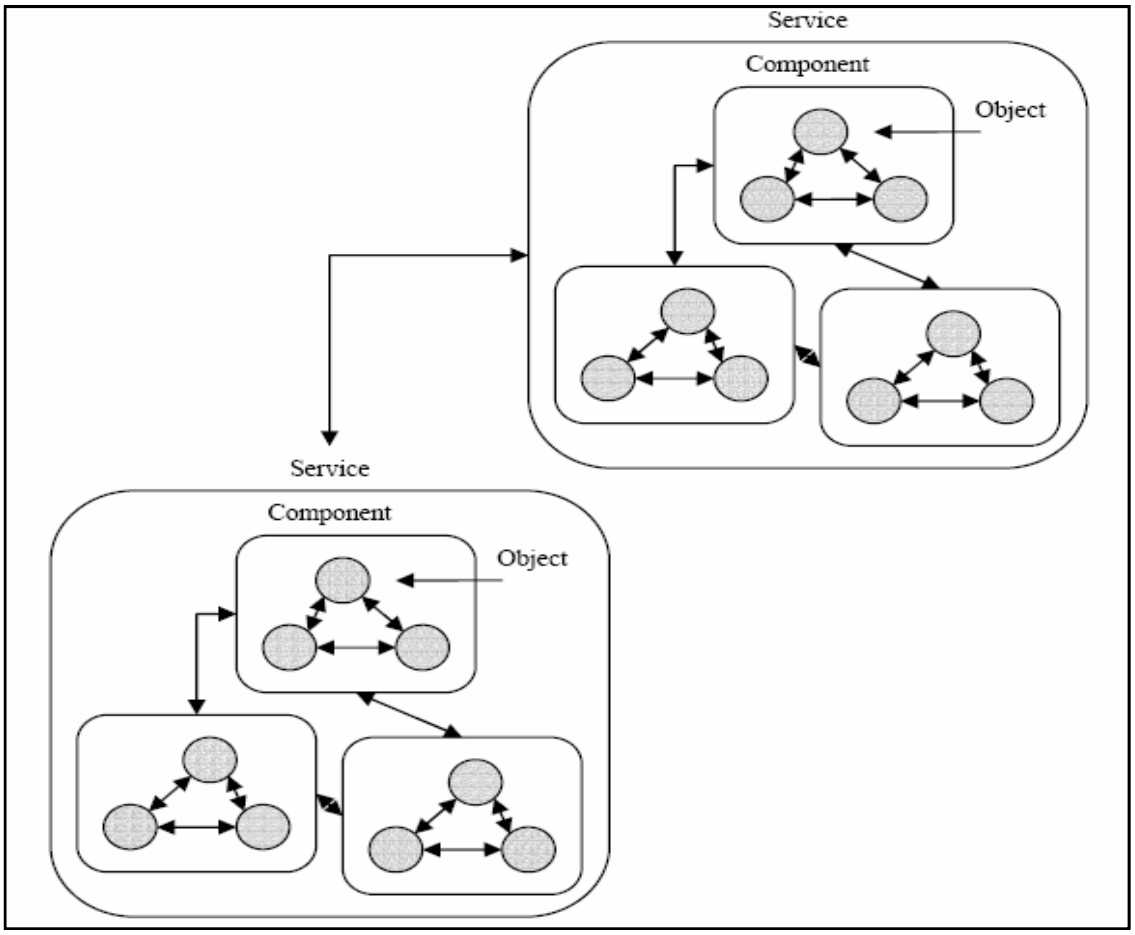

Figure 2.5. Service Based Development
(Source: Sehring 2006)

The early architecture of software is based on structured design, which has rigid rules for the development of software constructs and limited support to enable robust and sophisticated application development. Object oriented technologies result in flexible software development that supports encapsulation of business logic through more coarser-grained functions and classes; however, the tangible benefits of robust application development are gained through the progression of components and services. The Table 2.1 discusses the characteristics and features of each software architectural models.

Table 2.1. Comparison of Architectural Development Models
(Source: Sehring 2006)

Structured Development	Object Oriented Development	Component Based Development	Service Based Development
Very fine system structuring Low reusability Tight coupling Have compile time dependencies Intra-application communication scope	Small – grain system structuring Low reusability Tight coupling Have compile time dependencies Building blocks are individual classes Encapsulation, Inheritance, Polymorphism Functionality is described by class declarations Dynamic but large number of connected object	Medium – grain system structuring Medium reusability Loose coupling Have compile time dependencies Building blocks consist of several classes (components) Interactivity, connectivity, and exchangeability of components Functionality is described by interface declarations	Coarse – grain system structuring High reusability Loose coupling Have only run time dependencies Building blocks consist of components Published interface definition Dynamically discoverable distributed services Functionality is described by network addressable component interface declarations Inter-enterprise communication scope

2.1.3 SOA Layered Architecture

Currently the most frequently used application development model is based on three-tier architectural structure, which supports an additional layer between client and data storage tiers as shown Figure 2.6. The additional layer, called as business logic layer, provides code isolation from client and sharing of the application logic between various client implementations. It is a competent approach to software development for flexible managing of data and usage of system resources.

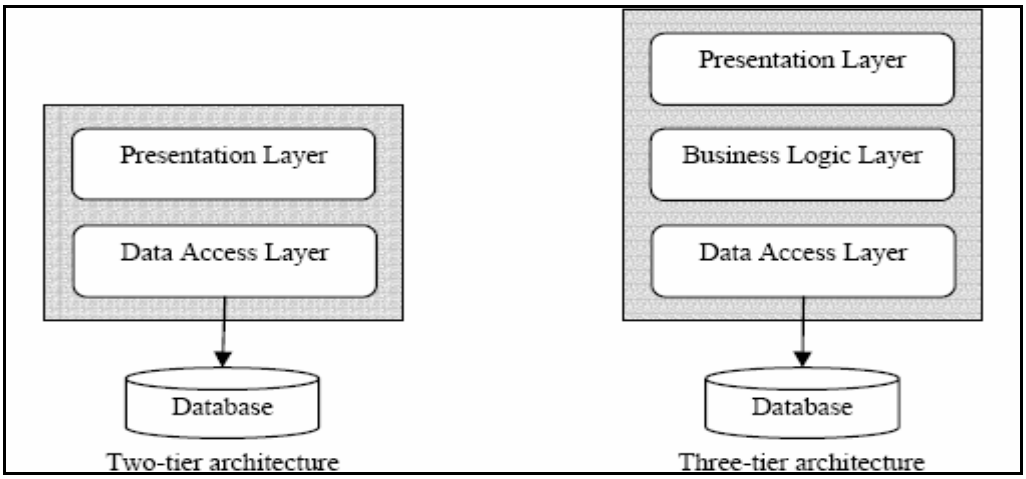

Figure 2.6. Two-tier and Three-tier Architectural Models
(Source: Sehring 2006)

SOA is based on n-tier application development in which services are layered on top of components that are responsible for providing certain functionalities and maintaining quality of service requirements for services (Sehring 2006), as shown in Figure 2.7.

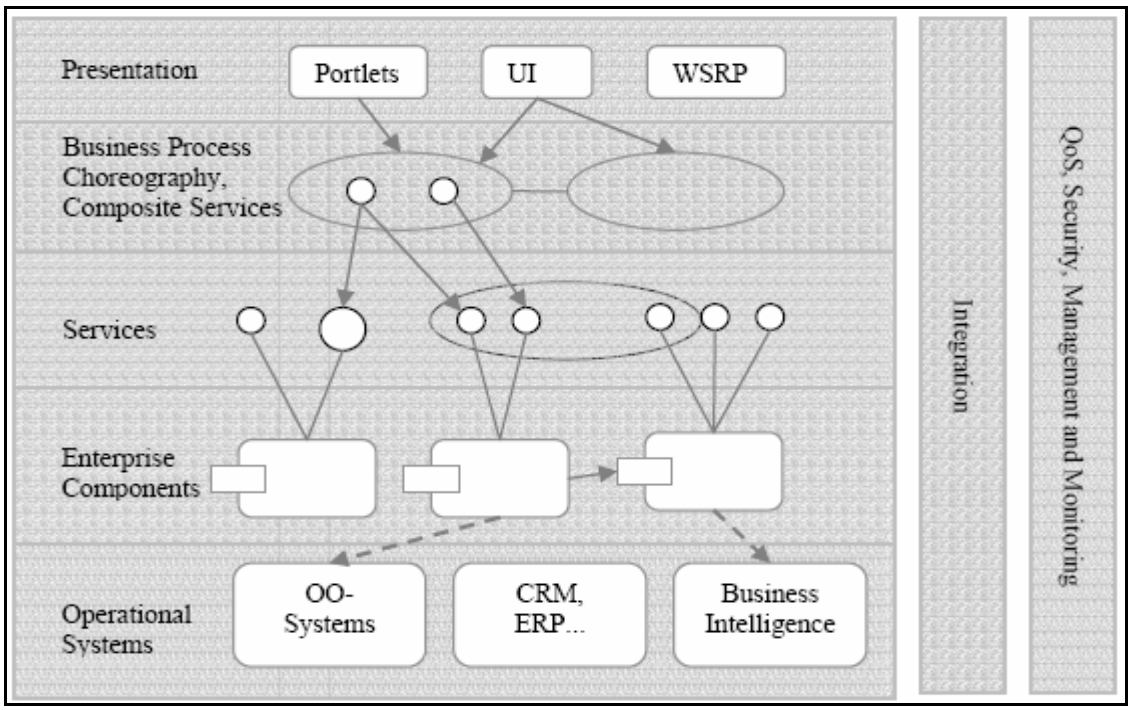

Figure 2.7. The Layers of Service Oriented Architecture
(Source: Sehring 2006)

2.2 Technologies for Service Oriented Architecture

The initial service-oriented technology was introduced in the late 1990s from Sun Microsystems, which is called as Jini Network Technology (Sun Microsystems 2001). Jini is a lightweight environment for dynamically discovering and using services on a network. Its main aim is to allow devices such as printers to dynamically connect to the network and register their available services.

Since Service Oriented Architecture (SOA) is an evolution of Object Oriented (component based architecture for separating functionality into individual objects that work together) and Distributed Systems such as J2EE, CORBA, and DCOM. Defining Web Services, .NET, J2EE and, CORBA as specialized SOA implementations that represent the core aspects of a service-oriented approach to service oriented architecture.

Web services are distributed software components that can be accessed through standard web protocols as shown Figure 2.8. The advantage of web services is that they can be consumed by any application that is able to parse and XML-formatted stream transmitted through HTTP. XML is the key technology in web services.

A web service is a component running on a web server that communicates to the world with standard Internet protocols such as: HTTP GET, HTTP POST, and SOAP (Simple Object Access Protocol).

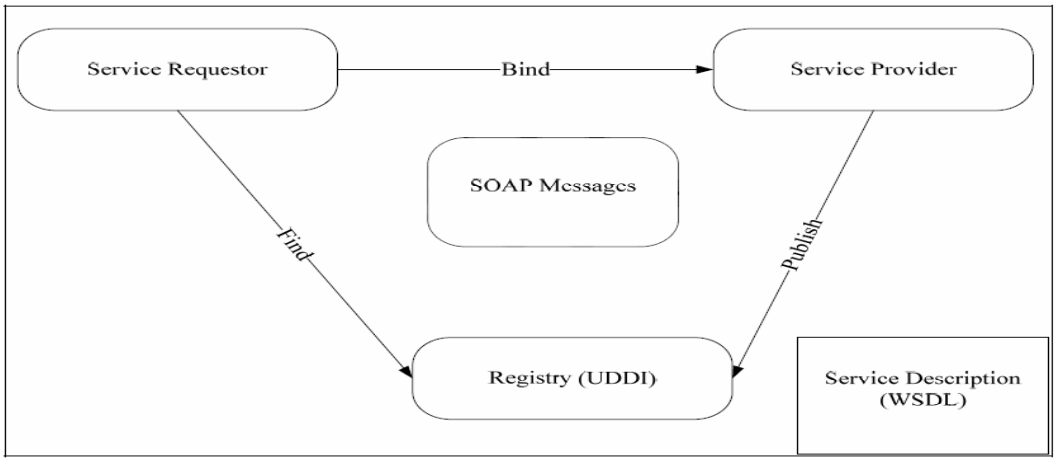

Figure 2.8. Web Service Architecture
(Source: Sehring 2006)

XML is a standard that provides encoding to data as plain text files. This has the significant advantage that almost anything can read it, as text files are standard across platforms. There are standards to adhere to, such as, what encoding language to use, and how much data is used for each character, but these are surmountable on all devices likely to be interested in web services (Microsoft Corporation 2002).

SOAP was originally designed as a means of manipulating objects remotely (W3Schools 2007). Now its use is more specialized, and is used almost exclusively with web service implementation. The two technologies looked at above, XML and XML schemas, are enough to exchange data with web services. The way two systems interact with each other across the Internet using SOAP is shown in Figure 2.9 (RSA Laboratories 2007).

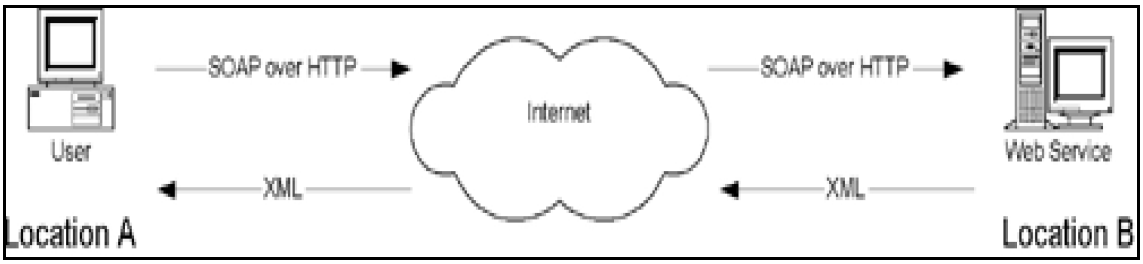

Figure 2.9. SOAP over HTTP
(Source: RSA Laboratories 2007)

Web services have an associated WSDL (Web Service Description Language) document describing all operations that a web service can perform. The description of a web service operation consists of the data structures used, the combinations of these data structures present in requests to and responses from the service, the format of messages containing these requests and responses, and the specific method of access required for the different operations that are supported by the service as shown Figure 2.10. In order to achieve this, WSDL documents which are written in XML contain schema information, information associating schema definitions with message structure, and some HTTP and SOAP specific information. Given a WSDL document it is possible to write code to access the web service it pertains to without any additional information (Bustos and Watson 2002).

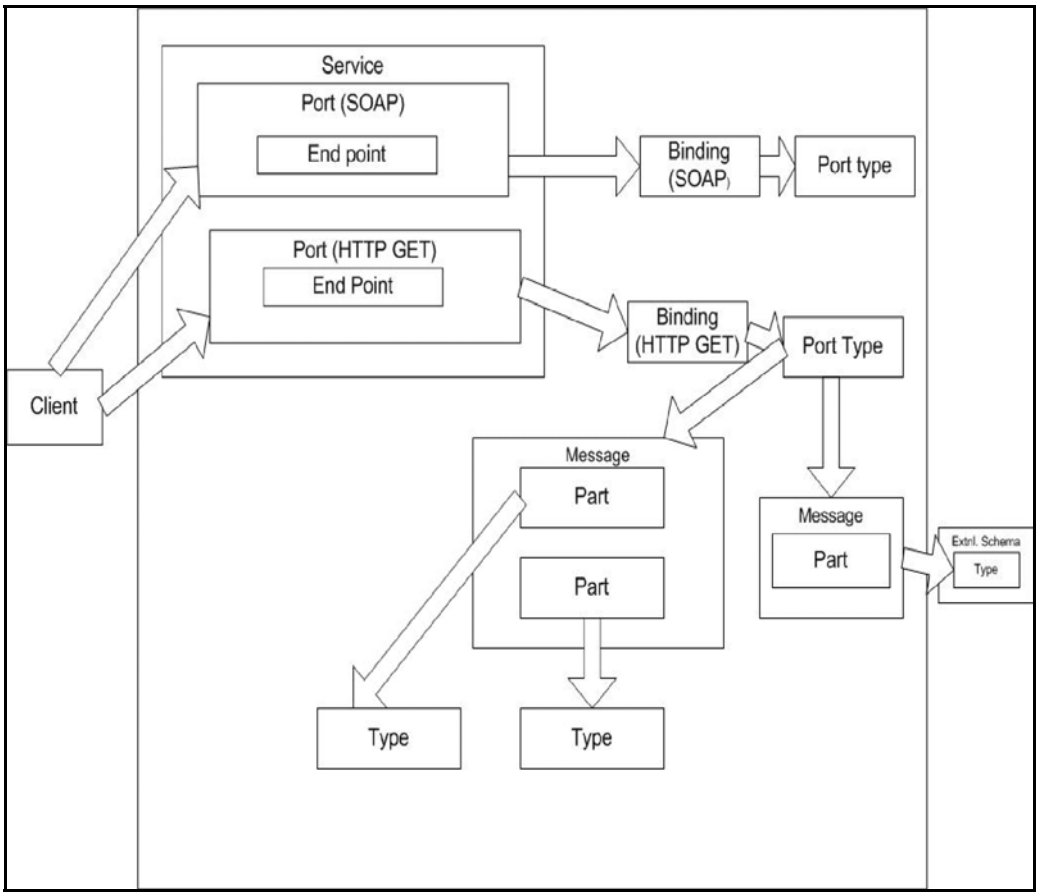

Figure 2.10. Schematic layout of WSDL
(Source: Bustos and Watson 2002)

A Proxy resides on the consumer's machine and acts as a relay between the consumer and the web service. When a proxy is built, it uses a WSDL file to create a map that tells the consumers what are the methods are available and how to call them (Birdwell and Cornes 2001). The consumer then calls the web method that is mapped in the proxy, which in turn, makes calls to the actual web service over the Internet. The proxy handles the entire network related work including sending of data, as well as managing the underlying WSDL.

WSDL accurately defines web services, but this is not sufficient to allow people to find the web service (Curbera and Duftler 2002). If the web site is not known, it will not be possible to find the WSDL that describes it. Some central access or a repository that people can query in order to find web services is needed and UDDI is used for this purpose.

2.3 Web Services of .NET Framework

All web services in .Net have the following elements (Thai and Lam 2000):

1. An *asmx* file for the web service. This must contain the `<% webservice ... %>` directive, as well as the class that provides the web service implementation. To the web wervice clients, this *asmx* file is the entry point to the web service. Instead of deploying as it is with Java Enterprise Edition (JEE) with .NET framework this file should be in a virtual directory with permits.
2. Inherit from the web service class of the System.Web.Services namespace. This allows the derived class to access all the normal ASP objects exposed in the web service base class.
3. Tag the methods that are going to be accessed on the web with WebMethod attributes.
4. A configuration file called *web.config* is placed in the same directory as the *asmx* file. This configuration file controls many settings from the virtual directory.

2.4 Security Mechanisms

Table 2.2 lists the security mechanisms defined in X.800. As can be seen from the list, the mechanisms are divided into those that are implemented in a specific protocol layer and those that are not specific to any particular protocol layer or security service. X.800 distinguishes between reversible encipherment mechanisms and irreversible encipherment mechanisms (Stallings 2003). A reversible encipherment mechanism is simply an encryption algorithm that allows data to be encrypted and subsequently decrypted. Irreversible encipherment mechanisms include hash algorithms and message authentication codes, which are used in digital signature and message authentication applications.

Table 2.2 X.800 Security Mechanisms
(Source: Stallings 2003)

SPECIFIC SECURITY MECHANISMS

May be incorporated into the appropriate protocol layer in order to provide some of the OSI security services.

Encipherment
The use of mathematical algorithms to transform data into a form that is not readily intelligible. The transformation and subsequent recovery of the data depend on an algorithm and zero or more encryption keys.

Digital Signature
Data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery (e.g., by the recipient).

Access Control
A variety of mechanisms that enforce access rights to resources.

Data Integrity
A variety of mechanisms used to assure the integrity of a data unit or stream of data units.

Authentication Exchange
A mechanism intended to ensure the identity of an entity by means of information exchange.

Traffic Padding
The insertion of bits into gaps in a data stream to frustrate traffic analysis attempts.

Routing Control
Enables selection of particular physically secure routes for certain data and allows routing changes, especially when a breach of security is suspected.

Notarization
The use of a trusted third party to assure certain properties of a data exchange.

PERVASIVE SECURITY MECHANISMS

Mechanisms that are not specific to any particular OSI security service or protocol layer.

Trusted Functionality
That which is perceived to be correct with respect to some criteria (e.g., as established by a security policy).

Security Label
The marking bound to a resource (which may be a data unit) that names or designates the security attributes of that resource.

Event Detection
Detection of security-relevant events.

Security Audit Trail
Data collected and potentially used to facilitate a security audit, which is an independent review and examination of system records and activities.

Security Recovery
Deals with requests from mechanisms, such as event handling and management functions, and takes recovery actions.

Table 2.3, based on one in X.800, indicates the relationship between security services and security mechanisms. X.800 defines a security service as a service provided by a protocol layer of communicating open systems, which ensures adequate security of the systems or of data transfers. A security service is a processing or communication service that is provided by a system to give a specific kind of protection to system resources; security services implement security policies and are implemented by security mechanisms (Stallings 2003).

Table 2.3 Relationship between Security Services and Mechanisms
(Source: Stallings 2003)

	Mechanism							
Service	Enciph-erment	Digital signature	Access control	Data integrity	Authenti-cation exchange	Traffic padding	Routing control	Notari-zation
Peer entity authentication	Y	Y			Y			
Data origin authentication	Y	Y						
Access control			Y					
Confidentiality	Y						Y	
Traffic flow confidentiality	Y					Y	Y	
Data integrity	Y	Y		Y				
Nonrepudiation		Y		Y				Y
Availability				Y	Y			

2.4.1 Symmetric Encryption and Message Confidentiality

The most commonly used symmetric encryption algorithms are block ciphers. A block cipher processes the plaintext input in fixed-size blocks and produces a block of cipher text of equal size for each plaintext block. This thesis focuses on the three most important symmetric block ciphers: the Data Encryption Standard (DES) and Triple DES (3DES), and Rijndael.

A symmetric encryption scheme has five ingredients shown on Figure 2.11.

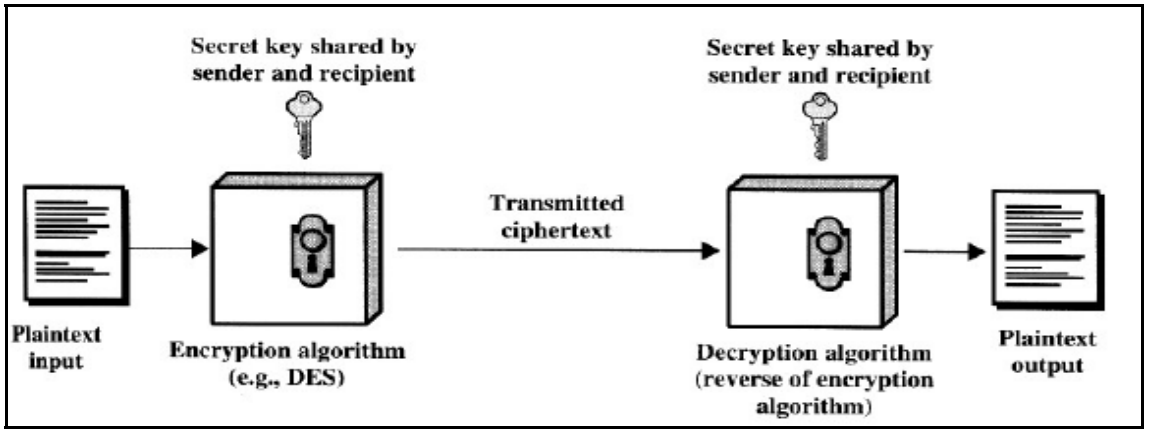

Figure 2.11. Simplified Model of Conventional Encryption
(Source: Stallings 2003)

The most powerful, and most common, approach to countering the threats to network security is encryption. In using encryption, we need to decide what to encrypt and where the encryption gear should be located. There are two fundamental alternatives; link encryption and end-to-end encryption; these are illustrated in use over a packet-switching network in Figure 2.12.

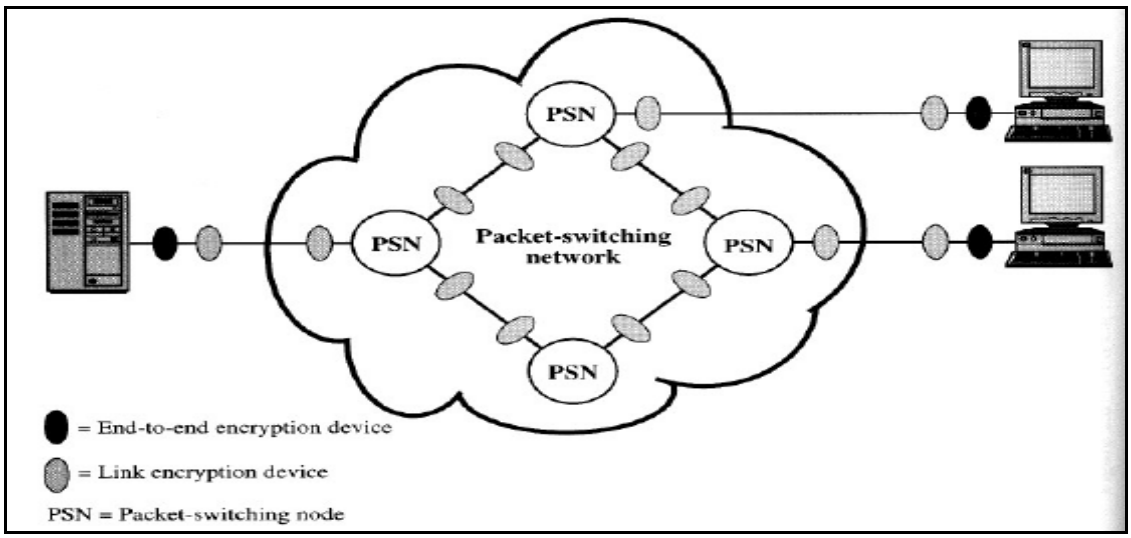

Figure 2.12. Encryption across a Packet-Switching Network
(Source: Stallings 2003)

One big issue with using symmetric algorithms is the key exchange problem. The other main issue is the problem of trust between two parties that share a secret symmetric key. Problems of trust may be encountered when encryption is used for authentication and integrity checking. A symmetric key can be used to verify the identity of the other communicating party, but this requires that one party trust the other (Thorsteinson and Ganesh 2003).

Key distribution can be achieved in a number of ways. For two parties A and B,

1. A key should he selected by A and physically delivered to B.
2. A third party could select the key and physically deliver to A and B.
3. If A and B have previously and recently used a key, one party could transmit the new key to the other, encrypted using the old key.
4. If A and B each have an encrypted connection to a third party C, C could deliver a key on the encrypted links to A and B.

2.4.2 Public Key Cryptography and Message Authentication

A message authentication code (MAC) is an authentication tag (also called a checksum) derived by applying an authentication scheme, together with a secret key, to a message. Unlike digital signatures, MACs are computed and verified with the same key, so that they can only be verified by the intended recipient. There are four types of MACs: unconditionally secure, hash function-based, stream cipher-based, or block cipher-based (RSA Laboratories 2007).

The use of cryptographic hash functions like MD5 or SHA for message authentication has become a standard approach in many Internet applications and protocols. Though very easy to implement, these mechanisms are usually based on ad hoc techniques that lack a sound security analysis (Bellare, et al. 1996).

The SHA-1 is called secure because it is computationally infeasible to find a message which corresponds to a given message digest, or to find two different messages which produce the same message digest. Any change to a message in transit will, with very high probability, result in a different message digest, and the signature will fail to verify. SHA-1 is a technical revision of SHA (FIPS 180) (Federal Information Processing Standards Publication 1995).

Public key cryptography, which uses a public key to encrypt the message and a private key to decrypt it. Public key systems are also known as asymmetric key cryptography. Public key cryptography is most often used for creating *digital signatures* on data, such as electronic mail, to certify the data's origin and integrity (Garfinkel and Spafford 1996).

2.4.3 .NET Cryptography

SymmetricAlgorithm has only one public constructor that takes no parameters. This constructor initializes the new instance with a randomly generated secret key. Of course, SymmetricAlgorithm also supports the standard methods Equals, Finalize, GetHashCode, ToString, GetType, and MemberwiseClone, which are defined in the base class Object. Figure 2.13 shows the symmetric algorithm class hierarchy.

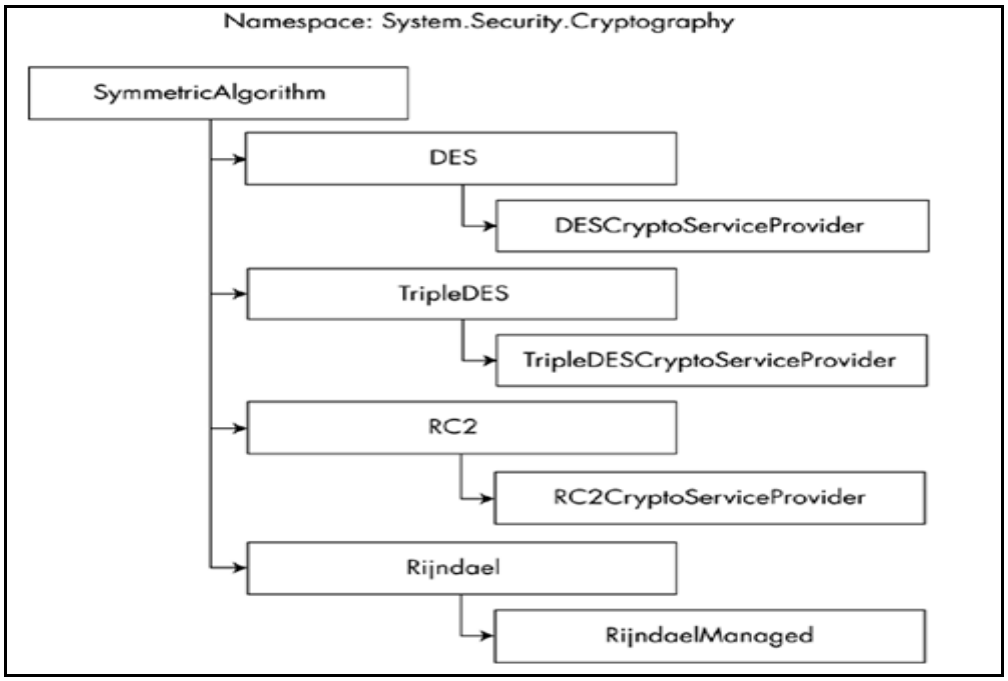

Figure 2.13. The symmetric algorithm hierarchy
(Source: Thorsteinson and Ganesh 2003)

AsymmetricAlgorithm has the similar methods and Figure 2.14 shows where AsymmetricAlgorithm class resides in the class hierarchy, under the abstract AsymmetricAlgorithm class.

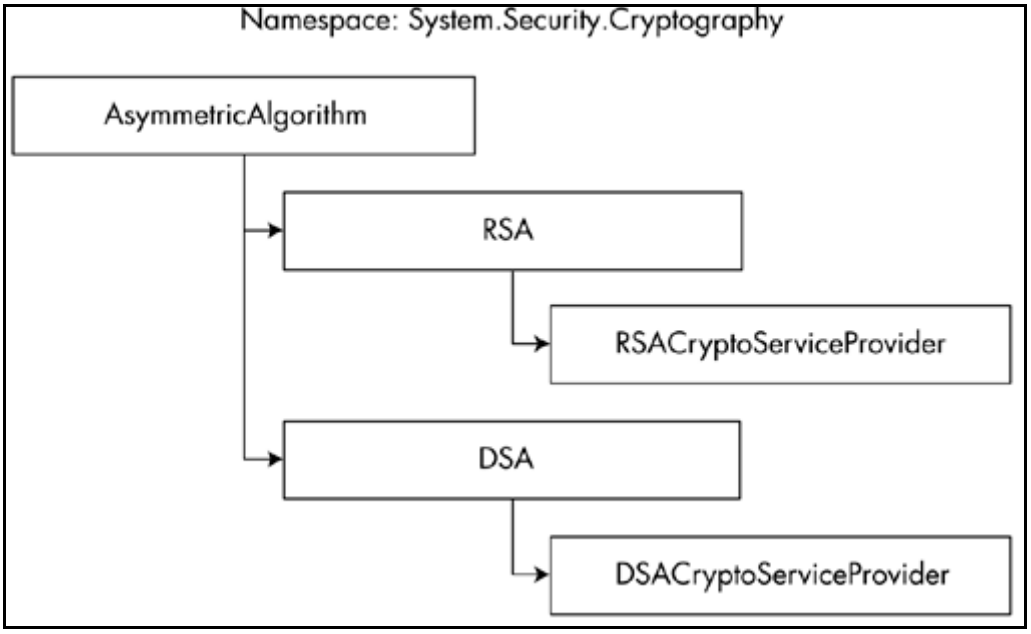

Figure 2.14. The asymmetric algorithm class hierarchy
(Source: Thorsteinson and Ganesh 2003)

For digital signatures, it turns out that it is not actually necessary to encrypt the entire original message. It is entirely sufficient, as well as much more efficient, to generate a hash of the original message, and then just encrypt that smaller hash value with the private key. Anyone with the matching public key (i.e., everybody) can then decrypt that hash with the public key for verification purposes. If the decrypted hash matches the recalculated hash of the actual message received, then the receiver can be quite confident that the original message that generated the hash must also match the received message. This comes from the fact that it is extremely difficult to find any two inputs that produce the same hash output (Thorsteinson and Ganesh 2003).

CHAPTER 3

WEB SERVICES SECURITY

3.1 Web Services Security Overview

There are seven requirements that must be addressed by a general security framework as defined by the ISO Security Standard (Wahli, et al. 2006):

1. Identification
2. Authentication
3. Authorization
4. Integrity
5. Confidentiality
6. Auditing
7. Non-repudiation

Web services security is one of the most important web services subjects. When using web services, similar security exposures exist as for other Internet, middleware-based applications and communications. To demonstrate the web services security exposures, we are going to explain several major risk factors for a system with no security. Most common security risks have been depicted in Figure 3.1.

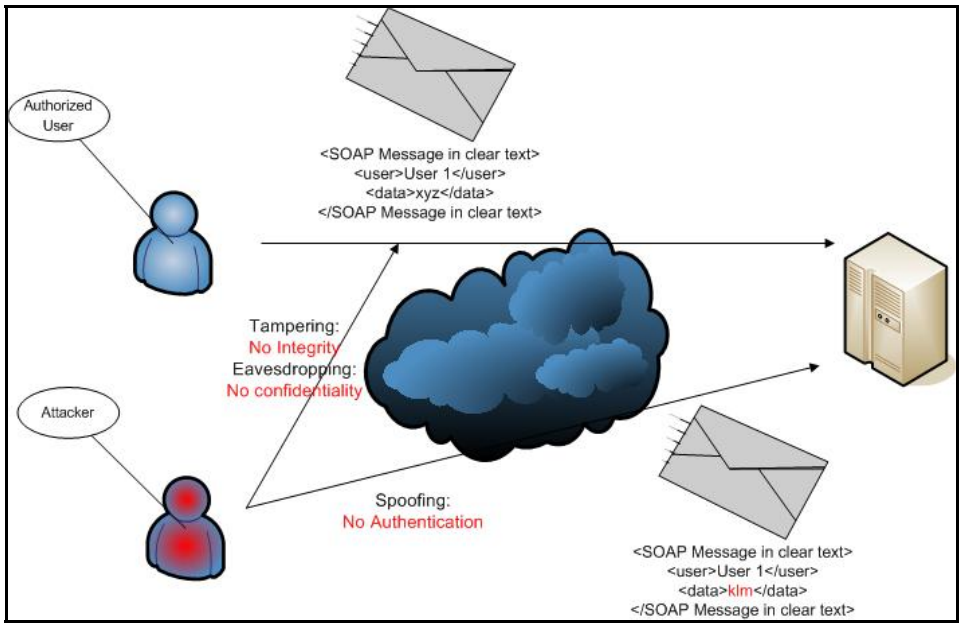

Figure 3.1. Web Services Scenario with No Security
(Source: Wahli, et al. 2006)

3.2 Web Services Security Approaches

Three fundamental concepts related to web services security exist from the perspective of web services architecture: the resources that must be secured, the mechanisms by which these resources are secured, and policies, which are machine-processable documents describing constraints on these resources.

Policies can be logically broken into two main types: permission policies and obligatory policies. A permission policy concerns those actions that an entity is permitted to perform and an obligatory policy concerns those actions that an entity is required to perform. Due to their nature, these two different kinds of policies have different types of enforcement mechanisms. A permission policy guard mechanism can be used to verify that a requested action is permitted to be performed by the entity, while the obligatory guard mechanism can only verify after the fact that an obligation has not been met. The architecture is principally concerned with the existence of such guard mechanisms and their role in the architecture.

Not all guards are active processes. For example, confidentiality of messages is provided by encryption. The guard here is the encryption itself, although this may be further backed up by active guards that apply policy.

Traditional network level security mechanisms such as Transport Layer Security (SSL/TLS), Virtual Private Networks (VPN), IPSec (Internet Protocol Security), and Secure Multipurpose Internet Mail Exchange (S/MIME) are point-to-point technologies. Although these traditional technologies may be used for web services security, they are not sufficient for providing an end-to-end security context, ad web services use a message oriented approach that enables complex interactions that can include the routing of messaged across various trust domains.

Therefore, message level security is important as opposed to point-to-point transport level security. As can be seen in Figure 3.2 below, the security context of SOAP message is end-to-end. However, there may also be a need for the intermediary to have access to some information in the message. This is illustrated as a security context between the intermediary and the original requester agent, and the intermediary and the ultimate receiver

You can send secured messages over many different protocols such as Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP), and Transmission Control Protocol (TCP) without having to rely on the protocol for security (Microsoft Patterns and Practices Developer Center 2005).

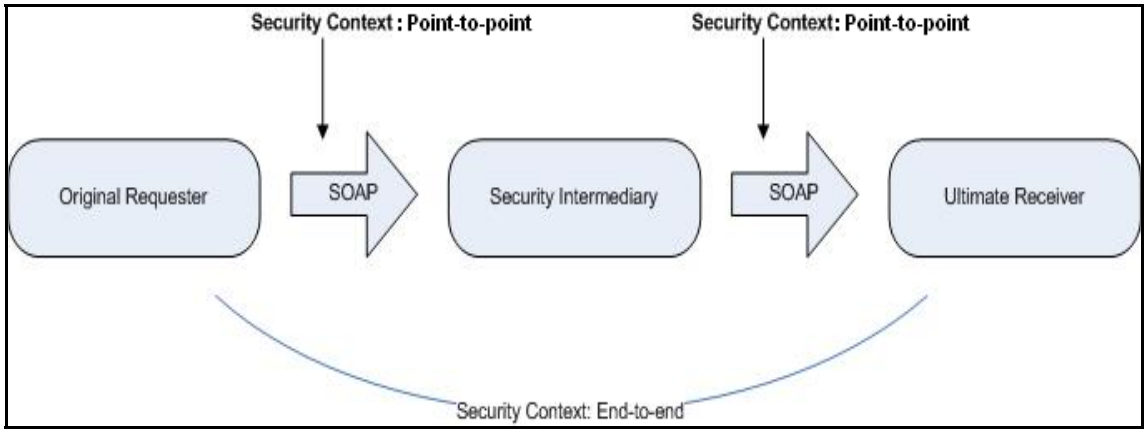

Figure 3.2. Point-to-point vs. End-to-end Security
(Source: W3C Working Group 2004)

HTTP, the most widely used Internet communication protocol, is currently also the most popular protocol for web services. HTTP is an inherently insecure protocol since all information is sent in clear text between unauthenticated peers over an insecure network. To secure HTTP transport-level security can be used. Transport-level security is a well known and often used mechanism for securing Internet and Intranet communications. It is based on Secure Sockets Layer or Transport Layer Security that works beneath HTTP.

If a message needs to go through multiple points to reach its destination, each intermediate point must forward the message over a new SSL connection. In this model, the original message from the client is not cryptographically protected on each intermediary because it traverses intermediate servers and additional computationally expensive cryptographic operations are performed for every new SSL connection that is established (Microsoft Patterns and Practices Developer Center 2005).

3.2.1 Web Services Security Technologies

In web services, the SOAP envelope is defined in XML, thus, web services can use many of the existing XML security technologies and standards, such as XML encryption and XML Digital Signatures. In addition many new standards such as WS-Security have emerged. The WS-Security is the cornerstone of all the efforts in pulling all these requirements together. The abstract of WS-Security specification document says that WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies (IBM Feeds 2002). Other technologies in the process of standardization are XML Key Management Specification (XKMS) (Ford, et al. 2001), Secure Assertion Markup Language (SAML) (Campbell and Lockhart 2007), Extensible Access Control Markup Language (XACML) (Parducci and Lockhart 2008), and Identity Federation (Raskin 2008).

3.2.2 WS-Security Overview

Multiple parts of a message can be secured in different ways. Multiple security requirements can be applied, such as integrity on the security token, and confidentiality on the SOAP body. End-to-end message level security can be provided through any number of intermediaries. WS-Security works across multiple transports and is independent of the underlying transport. Authentication of multiple party identities is possible.

The latest core specification, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) was standardized in March 2004. The two profiles, Web Services Security Username Token Profile, and Web Services Security X.509 Certificate Token Profile 1.0 were standardized at the same time. This evolution is presented in Figure 3.3.

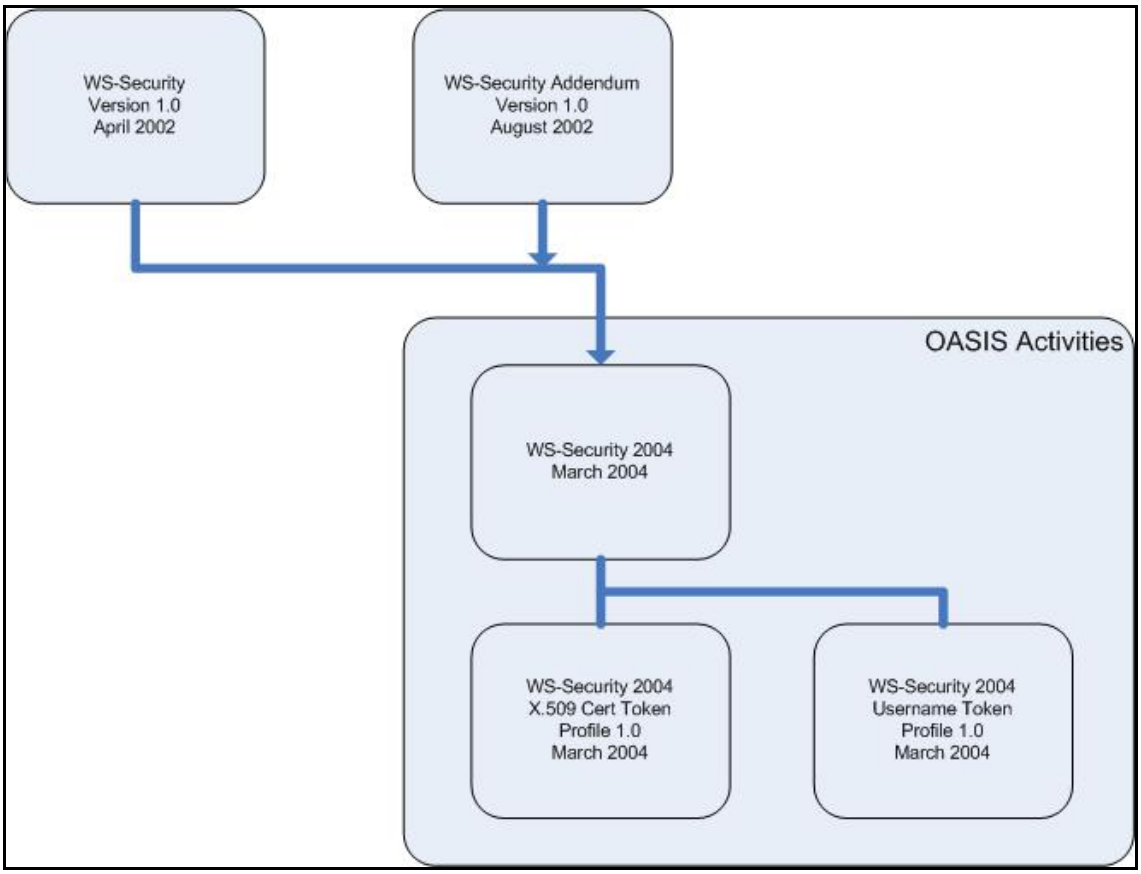

Figure 3.3. Evolution of WS-Security Specification
(Source: Wahli, et al. 2006)

The web services security model introduces a set of individual interrelated specifications to form a layering approach to security (Wahli, et al. 2006). This layered architecture is presented in Figure 3.4. It includes several aspects of security: identification, authentication, authorization, integrity, confidentiality, auditing, and non-repudiation.

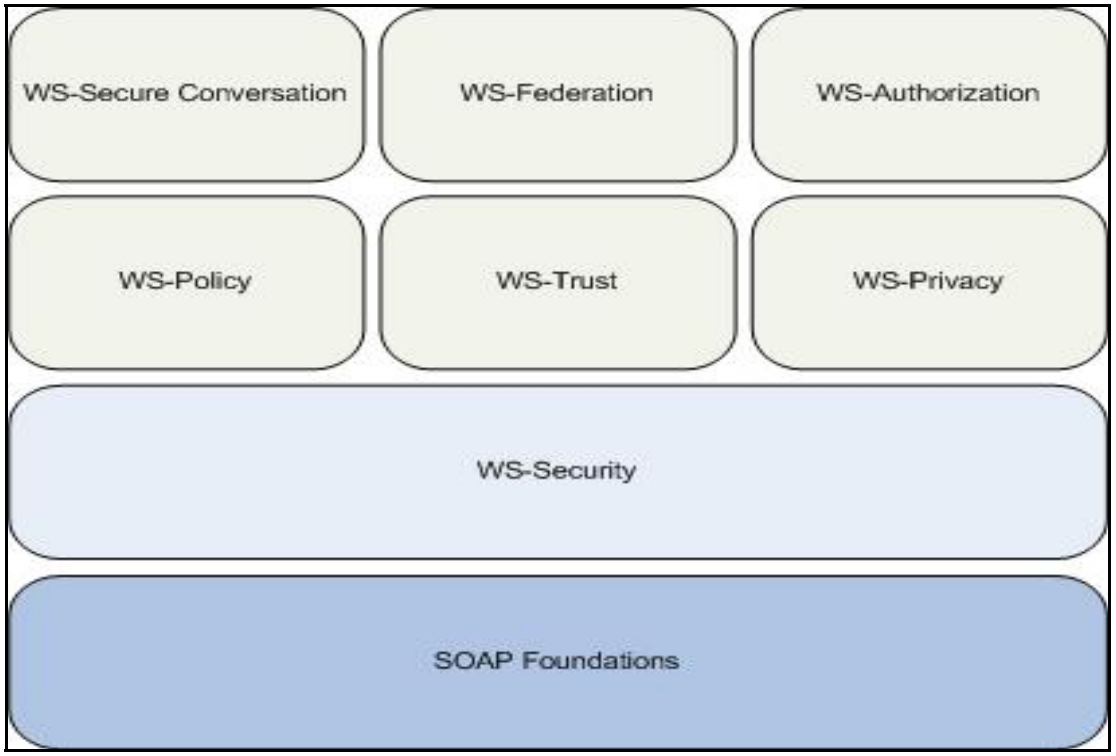

Figure 3.4. Web Services Security Specifications
(Source: Wahli, et al. 2006)

3.3 WS-Security Example

This section provides examples of SOAP messages with WS-Security. Using WS-Security, the authentication mechanism integrity, and confidentiality can be applied at the message level. As an overview, Figure 3.5 shows an example of web service security elements when the SOAP body is signed and encrypted.

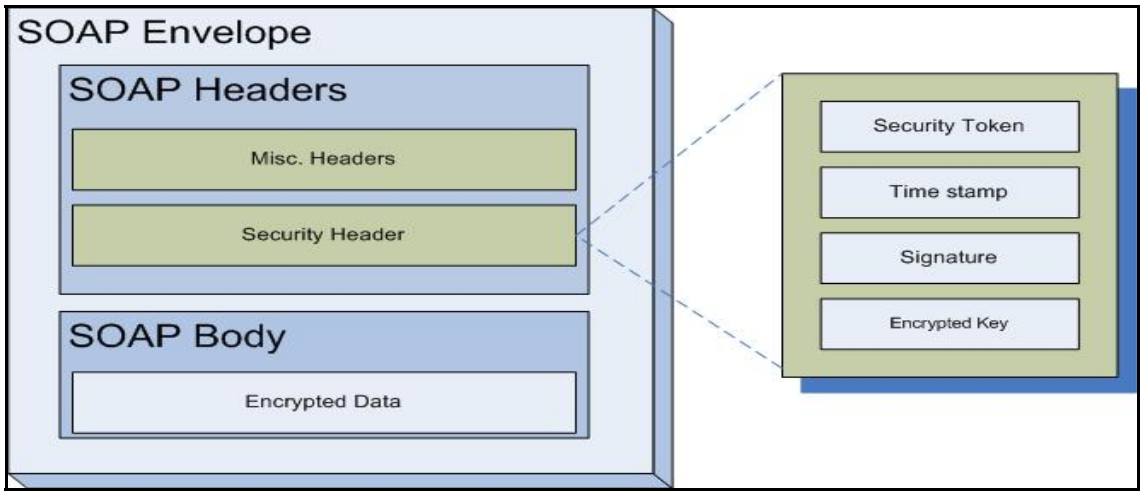

Figure 3.5. SOAP Message Security with WS-Security
(Source: Wahli, et al. 2006)

Figure A.1 shows the sample SOAP message without applying WS-Security. As you can see, there is only a SOAP body under the SOAP envelope. Applying WS-Security, the SOAP security header will be inserted under the SOAP envelope.

1. In Figure A.2, we show a message with authentication. As can be seen, we have username and password information as a <UsernameToken> tag in the message (Madsen 2006). When the username token is received by the web service server, the username and password are extracted and verified. Only when the username and password combination is valid will the message be accepted and processed at the server.

Integrity is applied to the application to ensure that no one illegally modifies the message while it is in transit. Essentially, integrity is provided by generating an XML digital signature on the contents of the SOAP message. If the message data changes illegally, the signature would no longer be valid.

Figure A.3 shows a sample SOAP message with integrity. Here the message body part is signed and added to the SOAP security header as signature information.

A signature is based on a key that the sender is authorized to have. Unauthorized sniffers do not have this key. When the receiver gets the message, it too creates a signature using message contents. Only if the two signatures match does the receiver honor the message. If the signatures are different, a SOAP fault is returned to the sender.

Figure A.4 shows a sample SOAP message with confidentiality. Here, the message body part is encrypted and a security header with encryption information added. Confidentiality is the process in which a SOAP message is protected so that only authorized recipients can read the SOAP message. Confidentiality is provided by encrypting the contents of the SOAP message using XML encryption. If the SOAP message is encrypted, only a service that knows the key can decrypt and read the message.

CHAPTER 4

WEB SERVICES SECURITY ARCHITECTURE

4.1 Web Services Security Architecture

We focus on WS-Security which is a basis for our implementation process on this thesis. Figure 4.1 is presented in order to understand WS-Security architecture. The processing of the architecture is performed as follows (Nakamura, et al. 2005):

1. Requester invokes WSSGenerator.
2. WSSGenerator invokes TokenGenerator to create a SecurityContextToken (SCT).
3. Since no SCT is cached initially, TokenGenerator invokes CallbackHandler.
4. CallbackHandler interacts with STS to get a Generic Service Security (GSS) Token and an SCT.
5. TokenGenerator stores a GSS token associated with an identifier of the SCT.
6. WSSGenerator invokes SigEngine, giving the SCT identifier.
7. During the sign operation, SigEngine eventually needs a key indicated by the SCT identifier, so KeyLocator is invoked.
8. KeyLocator finds a key with the identifier.
9. SigEngine signs the message, using the provided key.

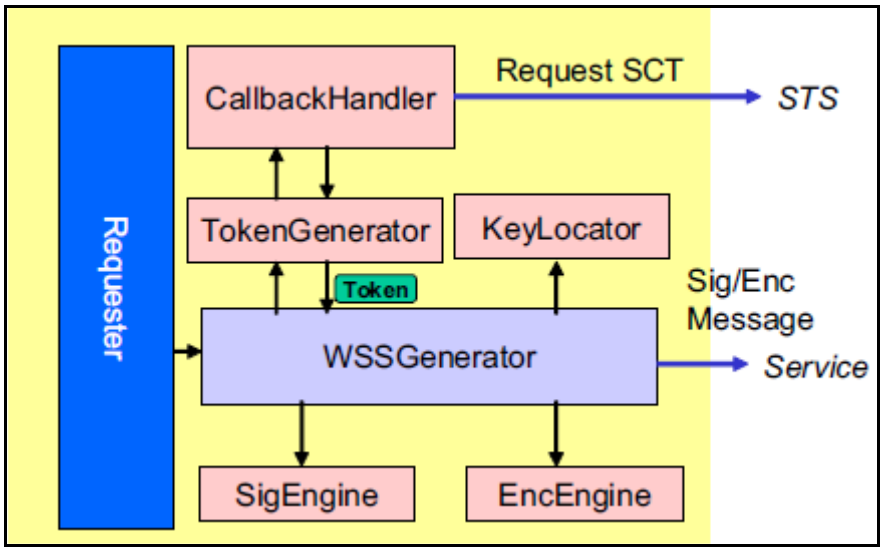

Figure 4.1. Architecture of WS-Security Handler in IBM WebSphere
(Source: Nakamura, et al. 2005)

On the other hand, when we focus on the implementation of the architecture the whole process explained step by step as follows:

1. Requester invokes WSSGenerator: The client class manages the generation process on the Client application.
2. WSSGenerator invokes TokenGenerator to create a SecurityContextToken (SCT): IDTokenGenerator and CreditTokenGenerator classes manage the token generation process.
3. Since no SCT is cached initially, TokenGenerator invokes CallbackHandler: Client class manages this process.
4. CallbackHandler interacts with STS to get a Generic Service Security (GSS) Token and an SCT: On STS side; token generator classes manages this process.
5. TokenGenerator stores a GSS token associated with an identifier of the SCT: This operation isn't implemented due to the performance analysis.
6. WSSGenerator invokes SigEngine, giving the SCT identifier: Signature Generation class manages that process.
7. During the signing, SigEngine eventually needs a key indicated by the SCT identifier, so KeyLocator is invoked: KeyExchange class manages key Exchange operation between client and STS. Client creates KeyExchangeToken object which includes SessionKey object encrypted with RSA algorithm with STS's

public key and sends to the STS. SignatureGeneration class operates signature process. SignatureClass signs the data with STS's private key.

8. KeyLocator finds a key with the identifier: STS's main class finds the Certificate stored on the local machine.
9. SigEngine signs the message, using the provided key: Certificate's public and private keys uses for signature operations.
10. For EncEngine process: Crypt interface is manages all symmetric encryption processes.

After the introduction of the web services security architecture in the following topic, a scenario based on web services security architecture will be explained.

4.2 Web Service Security Scenario

The scenario which will be used for demonstration of proposed architecture is a real-life example of the WS-Security scenario. The requester is a citizen who would like to apply for a credit account to a bank. For his application to be honored, the service requests the applicant to present two security tokens, one obtained from the government verifying his identity, and another token from the factious agency of Board of Credit Rating which rates people's credit status. When both tokens are acquired, the requester invokes the services web service and presents these tokens. The service verifies these tokens and according to the credit rating of the applicant, and decides to approve the credit request or not.

There are 6 steps of the scenario shown in Figure 4.2 (Hendrickson 2006):

1. The client requests an identification token from the identity security token service (IdentitySTS).
2. The identity security token service processes the request security token (RST) and sends a response.
3. The client requests a status token from the credit rating security token service (Credit Rating STS).
4. The Credit Rating STS processes the request status token and sends a response
5. The client sends a request to the service

6. The service processes the request, which includes identity security token and status token, and sends a response.

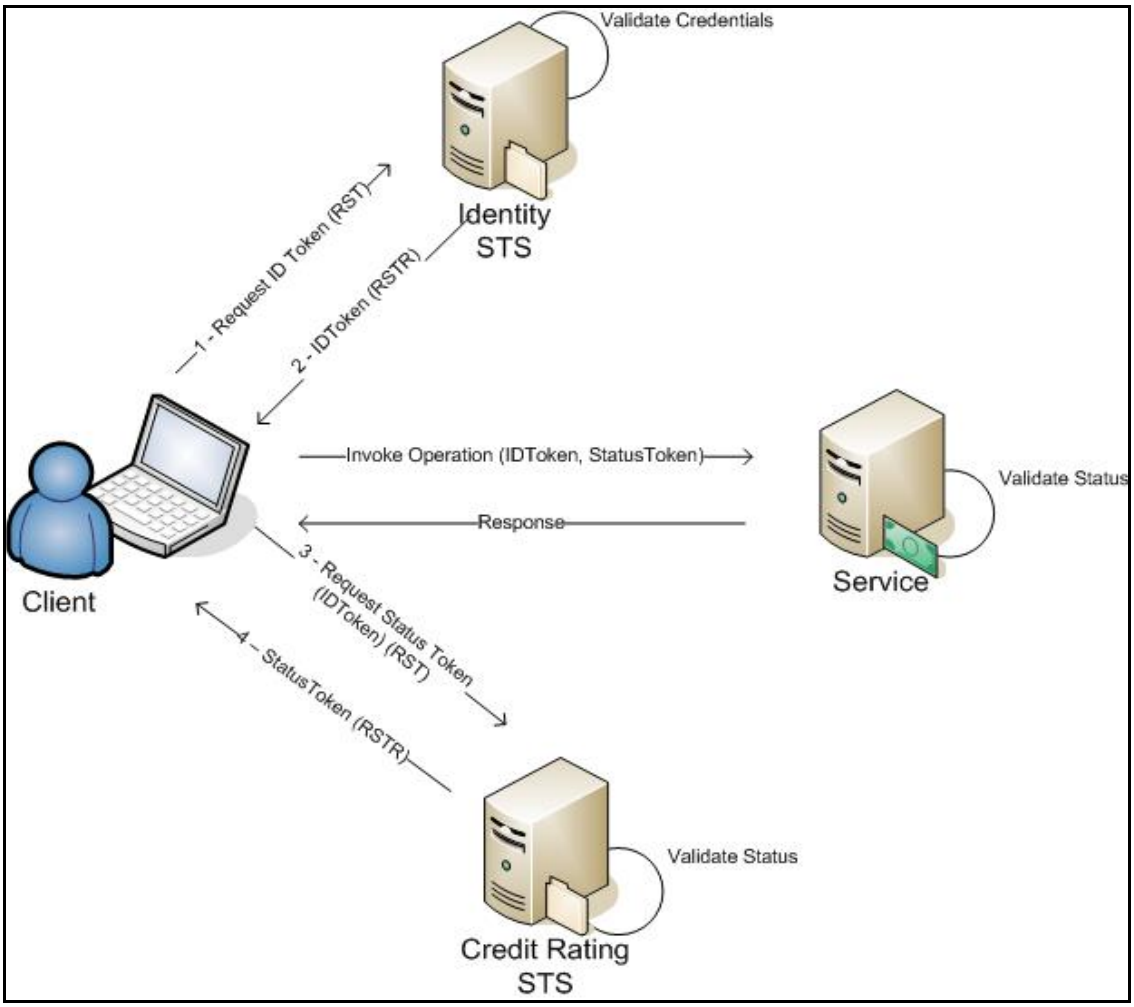

Figure 4.2. Scenario Flow
(Source: Hendrickson 2006)

The sequence diagram for the scenario flow is illustrated in Figure 4.3. Client application creates username, password information and transmits them to IdentitySTS web services. IdentitySTS web service uses information to verify the client and creates an identity token. Client application transmits the identity token, taken from IdentitySTS web service, to CreditRatingSTS web service to receive the credit status token. CreditRatingSTS uses identity token to determine the credit status and to create credit status token. Client application transmits the identity token and credit status token to service web service and service web service returns a response data after the verification processes of the tokens.

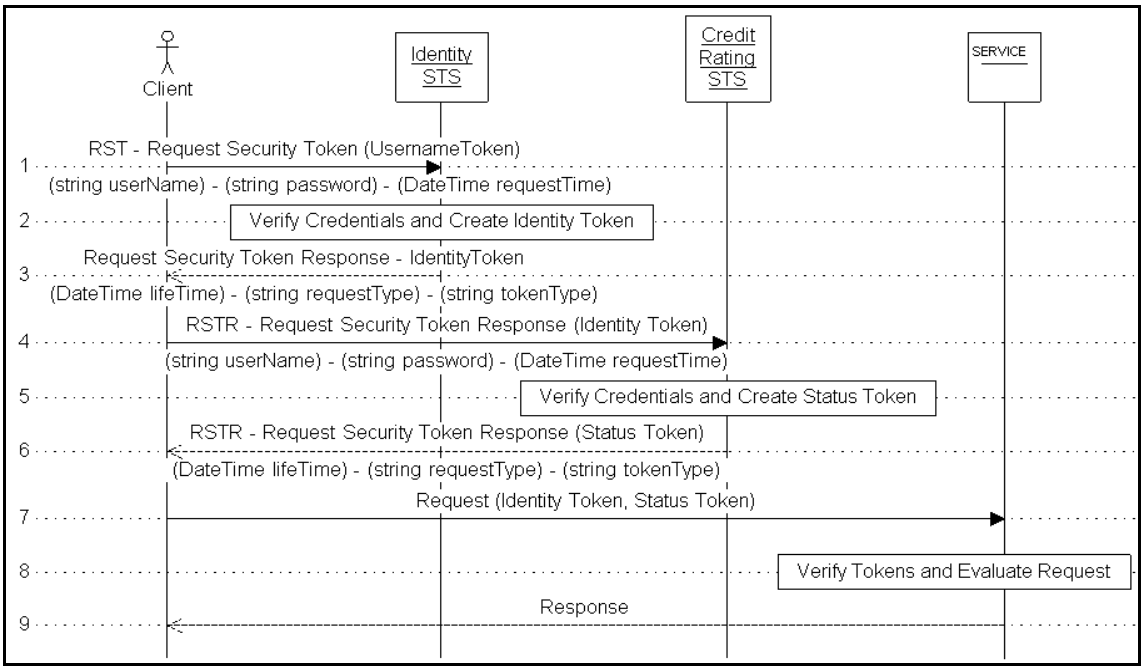

Figure 4.3. Sequence Diagram of the Scenario.

4.3 Overview of Security System of Scenario

The authentication security layer is provided between message level communication of client and IdentitySTS as shown Figure 4.4. Client creates username and password information. After the encryption approach with session key, client sends to the IdentitySTS to take an identity token behind the verification process by IdentitySTS.

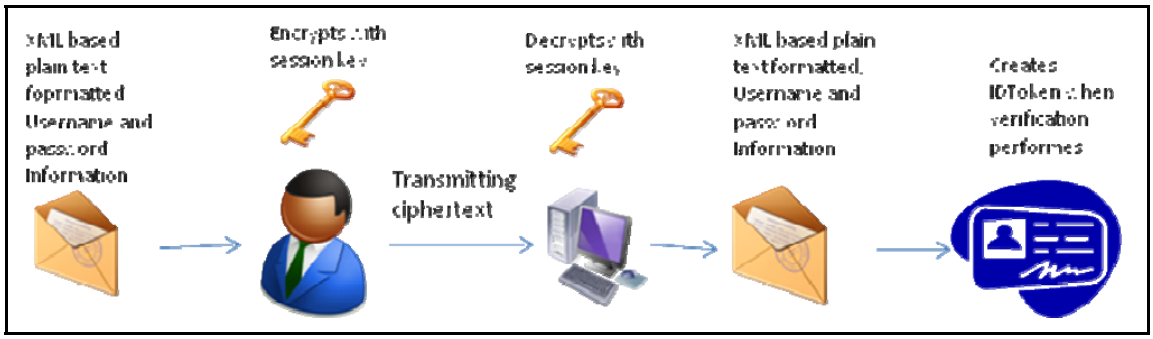

Figure 4.4. Authentication Process Flow

The integrity security layer is provided between client and STS's communication as shown Figure 4.5. After IDToken creation process of IdentitySTS, IdentitySTS firstly encrypts the IDToken with session key and signs the encrypted IDToken data through asymmetric private key of own certificate to perform the integrity security layer. When the client achieved the signed and encrypted IDToken sends to the CreditRatingSTS to obtain a CreditStatusToken. CreditRatingSTS first verifies the signature using with public key of the certificate of IdentitySTS. Behind verification complete, CreditRatingSTS creates a CreditStatusToken and encrypts the token with session key and signs the encrypted token data through asymmetric private key of own certificate to perform the integrity security layer. As the client achieved the signed and encrypted CreditStatusToken sends to the ServiceSTS to obtain an account. ServiceSTS verifies IDToken using the signature using with public key of the certificate of IdentitySTS and CreditStatusToken using the signature using through public key of the certificate of CreditRatingSTS. After verification process the service sends encrypted response.

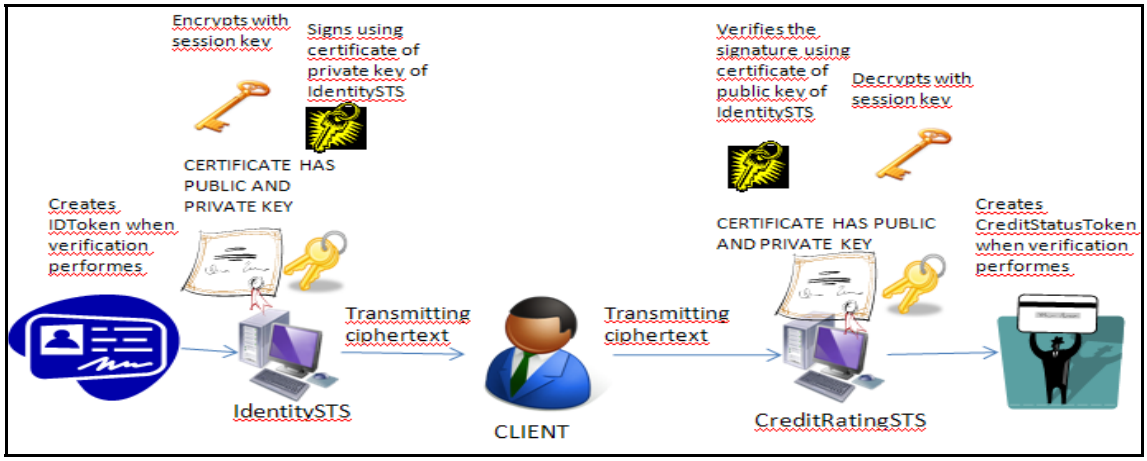

Figure 4.5. Integrity Process Flow

The confidentiality security layer is provided between client and STS's communication. All communications between client and STS's can be applied through a symmetric session key in cipher based. Session Key is selected by the user as Rijndael or Triple Des symmetric based algorithms. Session Key created by the client application. In order to perform this, as illustrated Figure 4.6, the session key is

performed into a key exchange method by using the asymmetric RSA algorithm. Ciphered session key is transmitted to STS's by an attribute of web method.

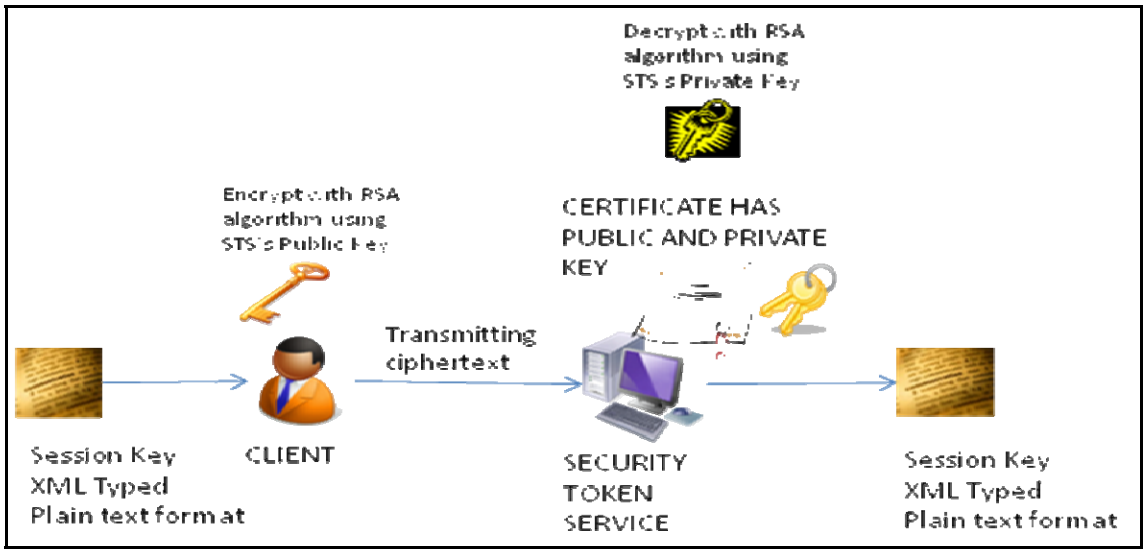

Figure 4.6. Key Exchange Flow

4.3.1 Identity Security Token Service Process

In the first step of the scenario, the client has two choices which are cryptographic based communication or plain text based communication. Client calls the requestIDToken method in order to communicate in the cryptographic based. However, client calls the requestIDTokenPlain method in order to communicate in the plain text based communication.

When the client calls the requestIDTokenPlain method, client sends a username token to the identity security token service (Identity STS) in plain text format. The identity STS is in charge of verifying credentials of the client using the username and password information. Identity STS verifies the username and password information by searching in a data store which involves the information about authorized clients. As seen on the Figure B.1, client calls the requestIDToken web method with the XML serialized ClientRequestInfos class type object attribute of the IdentitySTS web service in order to receive the identity token from the STS. The STS deserializes the XML serialized object and verifies the username and password information into the

ClientRequestInfos object by using check method of the IDCheck class which searches the data into the data store.

As the second step, according to the verification process, the identity security token service response the requested identity token which includes the type of requested token, the type of token and the lifetime of the requested token. In order to create the identity token, which is an object would be able to create from IDToken class, the STS calls the generate method of IDTokenGenerator class. Behind of the creation of the identity token, the STS performs the XML type serialization approach and returns the IDToken object to the client.

However, all these processes performed by the cryptographic based communication by the session key created by the client and sent to the IdentitySTS by the attribute of the requestIDToken web method.

4.3.2 Credit Security Token Service Process

In the third step of the scenario, the client again has two choices which are plain text or cryptographic based communication. Client calls the requestCreditStatus method in order to communicate in the cryptographic based but client calls the requestCreditStatus_PlainText method in order to communicate in the plain text based communication.

When the client calls the requestCreditStatus_PlainText method, client sends an identity token taken from IdentitySTS to the credit status security token service (Credit STS) in plain text format. The Credit STS is in charge of verifying credentials of the IdentitySTS token which has been signed by private key of the IdentitySTS. As seen on the Figure B.2, client calls the requestCreditStatus web method with the XML serialized CipherToken class type object attribute of the IdentitySTS web service in order to receive the credit status token from the STS.

As the forth step, according to the verification process, the credit status token service response the requested credit status token which includes the type of requested token, the type of token and the lifetime of the requested token. In order to create the credit status token, which is an object would be able to create from CreditStatusToken class, the STS calls the generate method of CreditStatusTokenGenerator class. Behind

of the creation of the credit status token, the STS performs the XML type serialization approach and returns the CreditStatusToken object to the client. However, all these processes performed by the cryptographic based communication by the session key created by the client and sent to the CreditSTS by the attribute of the requestCreditStatus web method.

4.3.3 Service Process

In the fifth step of the scenario, the client has two choices which are plain text or cryptographic based communication. Client calls the request method in order to communicate in the cryptographic based but client calls the request_Plain method in order to communicate in the plain text based communication.

When the client calls the request_Plain method, client sends identity token taken from IdentitySTS and credit status token taken from CreditSTS to the service in plain text format. The service is in charge of verifying credentials of the IdentitySTS token and CreditSTS token which has been signed by private key of the IdentitySTS and CreditSTS. As seen on the Figure B.3, client calls the request_Plain web method with the XML serialized CipherToken class type object attributes of the IdentitySTS web service and CreditSTS web service in order to receive the data from the service.

As the sixth step, according to the verification process, the service responses the requested data.

However, all these processes performed by the cryptographic based communication by the session key created by the client and sends to the service by the attribute of the request web method.

4.3.4 CipherToken Class Model

The communication, made between client and STS's, is based on ciphered data communication. The creation process for the cipher token first the data, which is a token (IDToken, CreditStatusToken), ciphered by a symmetric session key and stores into CipherData class. The creation times of the created tokens stores into the CreationTime

class which will be used for the statistical analysis. As shown Figure B.4, AuthenticationStatement class is utilized for storing the authentication time values which will be used for the statistical analysis and a pointer for KeyInfo class. CipherToken may also sign the ciphered data to ensure the integrity of the communication and the signature stores into the Signature class. The SignedInfo class which is pointed by the Signature class has all data for signature.

4.3.5 Domain Model

The domain model, represented on Figure B.5, designed for controlling all the messaging processes between client and web services. The Client calls the request method creates the three objects of ClientIdentitySTS, ClientCreditStatusSTS and ClientServiceSTS which are responsible for performing the communication by web services.

ClientIdentitySTS class has requestIDToken which firstly checks the cryptographic communication approach chosen by the user on the graphic user interface, secondly if plain text based communication is chosen then calls the IdentitySTSservices requestIDTokenPlain web method by the attribute of client information data else if creates symmetric session key by the chosen cryptographic method, thirdly ciphers the session key by using the RSA algorithm with IdentitySTSservices private key, fourthly ciphers the client information by the session key, fifthly calls IdentitySTSservice web services requestIDToken web method by the attributes of session key and client information data.

ClientCreditStatusSTS class has requestCipherToken_CreditStatusToken method which firstly checks the cryptographic communication approach chosen by the user on the graphic user interface, secondly if plain text based communication is chosen then calls the CreditStatusSTSservices requestCreditStatus_PlainText web method by the attribute of IDToken which received from IdentitySTSservice web service else if creates symmetric session key by the chosen cryptographic method, thirdly ciphers the session key by using the RSA algorithm with CreditStatusSTSservices private key, fourthly ciphers the IDToken by the session key, fifthly calls CreditStatusSTSservice

web services requestCreditStatus web method by the attributes of session key and IDToken data.

ClientServiceSTS class has requestCipherToken_Service method which firstly checks the cryptographic communication approach chosen by the user on the graphic user interface, secondly if plain text based communication is chosen then calls the Service web service's request_plain web method by the attribute of IDToken and CreditStatusToken which received from IdentitySTSservice and CreditStatusSTSservice web services else if creates symmetric session key by the chosen cryptographic method, thirdly ciphers the session key by using the RSA algorithm with Services private key, fourthly ciphers the IDToken and CreditStatusToken by the session key, fifthly calls Service web services request web method by the attributes of session key, IDToken and CreditStatusToken data.

4.3.6 Client Class Model

Except Client, ClientIdentitySTS, ClientCreditStatusSTS, ClientServiceSTS classes there are supporting classes as it is seen on the Figure B.6. Crypt interface manages the Cryptographic business to support the client. All token classes as IDToken, CreditStatusToken samples are also support client when XML serialization process. Client uses KeyExchange class to exchange the session key with web service by using the web services private keys. KeyExchangeToken has all information about session key and key Exchange information. Static type Serializer class operates the XML serialization approach between web services and client communication based on XML messaging system. XML messages converting the class object samples by using the Serializer class. Static type CryptAttribute class stores the information about cryptogragraphic keys and methods.

CHAPTER 5

IMPLEMENTATION AND EXPERIMENTS

5.1 Client Application

The client application, which requests tokens and demands a service from service security token service, has console application interface which presents to enter symmetric encryption type and test number options.

All time costs are saved to the files. All data will be used for performing statistical methods in order to test the communication type's efficiency.

All steps are described below for client application:

1. Create an instance of IdentitySTS web service.
2. Create a KeyExchange object to generate KeyExchangeToken object which contains sessionKeys encrypted with RSA algorithm utilizing with certificate of public key of IdentitySTS web service.
3. Perform an XML serialization operation for CipherToken object which encapsulates encrypted KeyExchangeToken and encrypted ClientRequestInfos objects with session key, and pointed by cipherValue field, in order to send IdentitySTS web service.
4. Request a CipherToken which encapsulates IDToken including digital signature from IdentitySTS web service by sending the attributes CipherToken objects encapsulates ClientRequestInfos and KeyExchange Token objects. Web method returns IDToken into the byte array encrypted by symmetric sessionKey algorithm.
5. Perform an XML deserialization operation for CipherToken object encapsulates IDToken object to achieve performance time costs.
6. Create an instance of CreditStatusSTS web service.

7. Perform XML serialization operation for CipherToken objects encapsulates encrypted KeyExchangeToken and encrypted IDToken objects, pointed by cipherValue field, in order to send CreditStatusSTS web service.
8. Request a CipherToken object encapsulates CreditStatusToken object including digital signature from CreditStatusSTS web service by sending the attributes CipherToken objects encapsulates IDToken and KeyExchangeToken objects. Web method returns CreditStatusToken into the byte array encrypted by symmetric sessionKey algorithm.
9. Create an instance of ServiceSTS web service.
10. Perform XML serialization operation CipherToken for KeyExchangeToken, CipherToken for IDToken, and CipherToken for CreditStatusToken in order to send to service web service.
11. Request a service encapsulated into CipherToken object including digital signature from ServiceSTS web service by sending the attributes IDToken, CreditStatusToken and KeyExchangeToken objects encapsulated into CipherToken object. Web method returns a response as affirmative or negatory into the byte array encrypted by symmetric sessionKey algorithm.

All communication between client and STSs has been provided by SOAP envelopes. An example SOAP message taken the time of requesting IDToken is illustrated on Figure C.1. The open message of ClientRequestInfos object pointed to the cipherValue field of CipherToken object may be shown on Figure C.3. An example SOAP message taken the time of requesting CreditStatusToken is presented on Figure C.5. An example SOAP message taken the time of requesting the service data is illustrated on Figure C.8.

5.2 Identity Security Token Services

IdentitySTS web service has two web methods to provide options for client to choose plain text based communication or secured based communication. RequestIDTokenPlain web method using for plain text based communication requires one attribute which is ClientRequestInfos encapsulated by CipherToken object on plain text format. RequestIDToken web method using for secure based communication

requires two attribute KeyExchangeToken object which has encrypted fields with asymmetric RSA algorithm and ClientRequestInfos encrypted with session key encapsulated by CipherToken object. Fundamentally, the structure of the web methods is very similar. Only the decryption, encryption and signature creation processes added to the requestIDToken web method. All steps are described below for requestIDToken web method of IdentitySTS:

1. Perform XML deserialization operation for Cipher Token object which encapsulates KeyExchangeToken object comes from client.
2. Decrypt KeyExchangToken object fields to obtain session key with RSA algorithm uses private key of certificate of IdentitySTS.
3. Create a SessionKey object which contains sessionKey information into plain text format. All information has been taken from decrypted KeyExchangeObject.
4. Decrypt CipherObject which encapsulates ClientRequestInfos object with session key.
5. Verify the identity information taken from decrypted ClientRequestInfos object using data store.
6. If verified create an IDToken.
7. Encrypt the IDToken with session key.
8. Sign the encrypted IDToken data using with private key of certificate of IdentitySTS.
9. Create a CipherToken object.
10. Include encrypted IDToken data to cipherValue field of CipherToken object
11. Include signature signatureValue field of CipherToken Object.
12. Add all the time cost values to the CipherToken object fields.
13. Return the CipherToken object.

An example SOAP message taken the time of responding IDToken is illustrated on Figure C.2. The open message of IDToken object pointed to the cipherValue field of CipherToken object may be shown on Figure C.4.

5.3. Credit Rating Security Token Services

CreditRatingSTS web service has two web methods to provide options for client to choose plain text based communication or secured based communication. requestCreditStatus_PlainText web method using for plain text based communication requires one attribute which is IDToken encapsulated by CipherToken object on plain text format. RequestCreditStatus web method using for secure based communication requires two attribute KeyExchangeToken object which has encrypted fields with asymmetric RSA algorithm and IDToken object encrypted with session key encapsulated by CipherToken object. Fundamentally, the structure of the web methods is very similar. Only the decryption, encryption and signature verification-creation processes added to the requestCreditStatus web method. All steps are described below for requestCreditStatus web method of IdentitySTS:

1. Perform XML deserialization operation for CipherToken object which encapsulates KeyExchangeToken object.
2. Decrypt KeyExchangeToken object fields to obtain session key with RSA algorithm uses private key of certificate of CreditRatingSTS.
3. Create a SessionKey object which contains sessionKey information into plain text format. All information has been taken from decrypted KeyExchangeObject.
4. Decrypt CipherObject which encapsulates encrypted IDToken object and its signature data with session key.
5. Verify the signature using with the public key of certificate of IdentitySTS and lifetime of the IDToken.
6. Create a CreditStatusToken if verified.
7. Encrypt the CreditStatusToken with session key.
8. Sign the encrypted CreditStatusToken data using with private key of certificate of CreditRatingSTS.
9. Create a CipherToken object.
10. Include encrypted CreditStatusToken data to cipherValue field of CipherToken object.
11. Include signature signatureValue field of CipherToken Object.

12. Add all the time cost values to the CipherToken object fields.
13. Return the CipherToken object.

An example SOAP message taken the time of responding CreditStatusToken is illustrated on Figure C.6. The open message of CreditStatusToken object pointed to the cipherValue field of CipherToken object may be shown on Figure C.7.

5.4 Service Security Token Services

ServiceSTS web service has two web methods to provide options for client to choose plain text based communication or secured based communication. Request_Plain web method using for plain text based communication requires two attribute which is IDToken encapsulated by CipherToken object on plain text format and CreditStatusToken encapsulated by CipherToken object on plain text format. Request web method using for secure based communication requires three attribute KeyExchangeToken objects which has encrypted fields with asymmetric RSA algorithm, IDToken object encrypted with session key encapsulated by CipherToken object and CreditStatusToken object encrypted with session key encapsulated by CipherToken object. Fundamentally, the structure of the web methods is very similar. Only the decryption, encryption and signature verification-creation processes added to the request web method. All steps are described below for request web method of ServiceSTS:

1. Perform XML deserialization operation for CipherToken object which encapsulates KeyExchangeToken object.
2. Decrypt KeyExchangToken object fields to obtain session key with RSA algorithm uses private key of certificate of ServiceSTS.
3. Create a SessionKey object which contains sessionKey information into plain text format. All information has been taken from decrypted KeyExchangeObject.
4. Decrypt CipherObject which encapsulates encrypted IDToken object and its signature data with session key.
5. Decrypt another CipherObject which encapsulates encrypted CreditStatusToken object and its signature data with session key.

6. Verify both signatures using with the public key of certificate of IdentitySTS, CreditStatusSTS and lifetimes of the IDToken, CreditStatusToken.
7. Create a service data if both verified.
8. Encrypt the service data with session key.
9. Sign the encrypted service data using with private key of certificate of ServiceSTS.
10. Create a CipherToken object.
11. Include encrypted service data to cipherValue field of CipherToken object.
12. Include signature signatureValue field of CipherToken Object.
13. Add all the time cost values to the CipherToken object fields.
14. Return the CipherToken object.

To sum up, all the processes of the implementation of our scenario has been explained step by step. In the next topic, experiments and its results will be presented in detail.

5.5 Experiments

In order to obtain operational values of distributed web services security architecture, a number of tests are performed using different cryptographic based algorithms. These tests are performed into the laboratory on four different computers the specifications of the computers written below:

1. Client Computer Specifications
    - Pentium 4 CPU 3.4 GHZ
    - 2.00 GB RAM
    - Windows XP SP2 Operating System
    - .Net Framework 2.0

2. IdentitySTS Computer Specifications

    - Pentium 4 CPU 2.00 GHZ
    - 512 MB RAM
    - Windows XP SP2 Operating System
    - .Net Framework 2.0

3. CreditStatusSTS Computer Specifications

    - Pentium 4 CPU 3.00 GHZ
    - 2.00 GB RAM
    - Windows XP SP2 Operating System
    - .Net Framework 2.0

4. ServiceSTS Computer Specifications

    - Pentium 4 CPU 3.40 GHZ
    - 2.00 GB RAM
    - Windows XP SP2 Operating System
    - .Net Framework 2.0

Service Security Services built into Internet Information Service 5.1 (IIS 5.1) on the top of .Net Framework 2.0 environment. Client application executes on the .Net Framework 2.0 environment. The results also affected operational system and .Net framework noises and background processes. Eliminating these problems is impossible. Due to that, evaluation of the performance results must be performed considering the background noise problem. The cryptographic algorithms specification is listed below:

1. Triple DES Algorithm
    - Triple DES BlockSize = 64
    - Triple DES FeedBackSize = 8
    - Triple DES Initialization Vector Length = 8
    - Triple DES Key Size = 192
    - Triple DES Mode = CBC
    - Triple DES Padding Mode = PKCS7
2. Rijndael Algorithm
    - Rijndael BlockSize = 128
    - Rijndael FeedBackSize = 128
    - Rijndael Initialization Vector Length = 16
    - Rijndael Key Size = 256
    - Rijndael Mode = CBC
    - Rijndael Padding Mode = PKCS7
3. RSA Algorithm
    - RSA KeyExchangeAlgorithm = RSA-PKCS1-KeyEx
    - RSA KeySize = 1024
    - RSA SignatureAlgorithm = http://www.w3.org/2000/09/xmldsig#rsa-sha1

The target is the observation statistically of Rijndael, Triple DES communications measurements on the .Net Framework 2.0 in different network settings. In order to obtain the time cost of each cycles test samples have been taken for each cryptographic based communication algorithms.

In order to observe the network settings effects three different network location has been used which are local, distributed and two computer network locations. In local structure all service and client application has been set up to client computer. In distributed structure all services and client application has been set up own machine. In two computer structure is all service has been set up to CreditStatusSTS computer and client application has been set up to client computer.

5.6 Evaluation of the Experiments in Statistical Methods

The analysis of variance (ANOVA) (Montgomery 1997) has been utilized for statistically evaluation of the data of the experiments. Analysis of variance is a statistical technique for analyzing experimental data. It subdivides the total variation of a data set into meaningful component parts associated with specific sources of variation in order to test a hypothesis on the parameters of the model.

16 replicated 2*3 factorial experiment (Montgomery 1997), which has symmetric cryptographic methods (Rijndael and Triple DES) and network settings (Local, Distributed and Two Computer) are factors, has been used. In this design, the symmetric cryptographic methods factor which is Rijndael and Triple DES is two level and the network settings factor consists of three level (Local, Distributed and Two Computer). All data has been obtained by calculating the means of 500 measurements.

The aim of the choice of factorial design is to determine the interactions between the factors which are to designate the effects. All experiments, which have been performed four times a day, have been consummated in four days (08-11.09.2008).

The ANOVA table shown in Table 5.1, which has been obtained complete analyses of variance model, is below. The model involving symmetric cryptographic methods, network settings and interaction effects. This model has been determined as statistically significant model ($F = 32.19$, $P < 0.0001$).

Table 5.1. ANOVA Table of Complete Analyses of Variance Model

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	20	0.48137131	0.02406857	32.19	<.0001
Error	75	0.05607238	0.00074763		
Corrected Total	95	0.53744369			

Means of main effects of response times which has been included in the model presented on Table 5.2, Table 5.3 and Table 5.4. After the analysis of variance, network settings effect included in model has been determined as statistically significant ($F=103.86$, $P<0.0001$). If time effects considered as block effect, time effects has been determined as statistically significant ($F = 29.05$, $P < 0.0001$). Symmetric cryptographic effect has been determined as statistically insignificant ($F= 0.29$, $P = 0.5899$). The interaction of symmetric cryptographic methods and network settings has been determined as statistically insignificant ($F = 0.06$, $P = 0.9448$) as shown on Table 5.5. Statistically insignificant interaction affects exhibits that symmetric cryptographic methods (Rijndael and Triple DES) react as independently through network settings (Local, Distributed and Two computer) vice versa.

Table 5.2. Means of Network Settings Effects

Level of NETWORK SETTINGS	N	score	
		Mean	Std Dev
2 Computer	32	0.44743359	0.05480471
Distributed	32	0.50733984	0.05762563
Local	32	0.54511914	0.07748063

Table 5.3. Means of Symmetric Cryptographic Methods Effects

Level of Cryptographic Methods	N	score	
		Mean	Std Dev
Rijndael	48	0.49845378	0.07529177
Triple DES	48	0.50147461	0.07590430

Table 5.4. Means of Experiment Repetitions

Level of Repetition	N	score	
		Mean	Std Dev
1	6	0.41628125	0.03436473
2	6	0.41517708	0.03352135
3	6	0.42054688	0.03689504
4	6	0.47943229	0.12119088
5	6	0.58668229	0.06460875
6	6	0.58857813	0.05572307
7	6	0.59486979	0.06464872
8	6	0.58821354	0.05668958
9	6	0.49957292	0.02650188
10	6	0.48925000	0.03510945
11	6	0.49131250	0.02896885
12	6	0.48823438	0.03706502
13	6	0.48945313	0.03684678
14	6	0.48391667	0.03262721
15	6	0.48425000	0.04101718
16	6	0.48365625	0.03248284

Table 5.5. The Analysis of Variance Table For Each Effects

Source	DF	Type I SS	Mean Square	F Value	Pr > F
NETWORK SETTINGS	2	0.15529067	0.07764533	103.86	<.0001
Cryptographic Method	1	0.00021901	0.00021901	0.29	0.5899
Repetition	15	0.32577668	0.02171845	29.05	<.0001
Network Settings* Cryptographic Method	2	0.00008495	0.00004247	0.06	0.9448

After the analysis of variance, Tukey's Studentized Range (HSD) Test has been used to compare the levels of network settings effect determined as statistically significant as shown on Table 5.6. The statistically significant difference has been determined for all network settings levels.

Table 5.6. Tukey's Studentized Range (HSD) Test To Compare The Levels of Network Settings

Comparisons significant at the 0.05 level are indicated by *.**				
Network Settings Comparison	**Difference Between Means**	**Simultaneous 95% Confidence Limits**		
Local - Distributed	0.037779	0.021434	0.054124	***
Local - 2 Computer	0.097686	0.081340	0.114031	***
Distributed – Local	-0.037779	-0.054124	-0.021434	***
Distributed – 2 Computer	0.059906	0.043561	0.076251	***
2 Computer – Local	-0.097686	-0.114031	-0.081340	***
2 Computer - Distributed	-0.059906	-0.076251	-0.043561	***

At the beginning of the experiments, the expected experiment results are definition of difference between network settings effect and symmetric cryptographic methods effect considering interaction with each other. Although expected results were determination of the statistically significant difference between symmetric cryptographic methods effect, statistical analysis couldn't discover the statistically significant effect for cryptographic methods Rijndael and Triple DES and interaction between network settings. The expected result for the network settings has been observed since the network settings effect has been determined statistically significant. As illustrated on Table 5.2, while considering the response times of our implementation of development of a web services security architecture based on .Net framework, the

shortest response time belongs to two computer network setting the second belongs to distributed network setting and the longest response time belongs to local network setting. These results must be commented considering the background noise of .Net Framework 2.0 and Windows XP since it is impossible eliminating and forecasting the noise of these platforms.

CHAPTER 6

CONCLUSION

In this thesis, the foundations for a more comprehensive framework for securing web services have been laid. The suggested architecture demonstrates an example of trust brokering for various purposes. The fact that each of the actors is located in different security domains can be considered as a hint for implementing a federation framework for service oriented architectures.

The Scenario have been chosen by considering four fundamental ingredients, which are authentication, authorization, integrity and confidentiality, of distributed web services security architecture. This scenario is implemented on the .NET Framework.

Web Service Security Architecture application only runs with tokens by using cryptographic algorithms on .NET Framework 2.0. The service verifies the tokens are valid or not and response a message by considering the tokens validation process. All processes time costs calculated and saved on a file to comment the efficiency and performance of the security ingredients cost of the architecture. 16 replicated 2*3 factorial experiment (Montgomery 1997), which has symmetric cryptographic methods (Rijndael and Triple DES) and network settings (Local, Distributed and Two Computer) are factors, has been used. In this design, the symmetric cryptographic methods factor which are Rijndael and Triple DES is two level and the network settings factor consists of three level (Local, Distributed and Two Computer). All data has been obtained by calculating the means of 500 measurements. Statistical analysis couldn't discover the statistically significant effect for cryptographic methods Rijndael and Triple DES and interaction among network settings. While considering the response times of our implementation of development of web service security architecture based on .Net framework, the shortest response time belongs to two computer network setting the followed by distributed network setting and the longest response time belongs to local network setting. These results must be commented considering the background noise of .Net Framework 2.0 and Windows XP since it is impossible eliminating and forecasting the noise of these platforms.

As a future work, other extensions such as support for different security tokens like Kerberos tokens can be added to this architecture. In order to discover the shortest response time, the properties of symmetric algorithms (Rijndael and Triple DES) may also be modified and a new experiment design can be performed for these new properties. For more robust key exchange operation, the properties of asymmetric algorithm (RSA) may be modified and the response times may be analyzed statistically. Background noises, which are impossible to eliminate, may be deeply analyzed statistically to find a function or a period to understand the attitude of the effects.

REFERENCES

Atkinson B., Libera M., Hada M. and Hondo I. 2002. Web Services Security (WS-Security), IBM, Microsoft, VeriSign, http://www-106.ibm.com/developerworks/webservices/library/ws-secure/ (accessed August 19, 2007).

Bass L., Clements P., Kazman R. 1997. *Software Architecture in Practice*. Addision-Wesley.

Bellare M., Canetti R. and Krawczyk H. 1996. Keying Hash Functions for Message Authentication. http://www-cse.ucsd.edu/~mihir/papers/kmd5.pdf (accessed February 5, 2008).

Birdwell R., Cornes O. 2001. *Beginning ASP. NET using VB.NET*. Wrox Press.

Booth D., Haas H. 2004. World Wide Web Consortium (W3C Working Goup). http://www.w3.org/TR/ws-arch/ (accessed November 22, 2007).

Bustos J. and Watson K. 2002. *Beginning .NET Web Services using Visual Basic.NET*. Wrox Press Ltd.

Campbell B., Lockhart H. 2007. OASIS Security Services (SAML), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security (accessed September 30, 2007).

Curbera F., Duftler M. 2002. *Unraveling web services web*. IEEE, Internet Computing.

Federal Information Processing Standards Publication 1995. Secure Hash Standard. http://www.itl.nist.gov/fipspubs/fip180-1.htm (accessed January 4, 2008).

Ford W., Hallam-Baker P., Fox B., Dillaway B., LaMacchia B., Epstein J. and Lapp J. 2001. XML Key Management Specification (XKMS). http://www.w3.org/TR/xkms/ (accessed February 28, 2008).

Garfinkel S., Spafford G. 1996. *Practical Unix and Security*. O'Reilly, 2nd edition.

Hasan J. 2006. *Expert Service-Oriented Architecture in C#2005*. Springer-Verlag New York, Inc.

Hendrickson S. 2006. *Web Services Security a Proposed Architecture For Interdomain Trust Relationship*. IYTE Master of Science.

IBM Feeds 2002. Web Services Security. http://www.ibm.com/developerworks/library/specification/ws-secure/ (accessed December 24, 2007).

Magedanz T., Blum N., Dutkowski S. 2007. Evolution of SOA Concepts in Telecommunications, http://doi.ieeecomputersociety.org/10.1109/MC.2007.384 (accessed February 15, 2008).

Madsen P. 2006. WS-Trust: Interoperable Security for Web Services. http://webservices.xml.com/pub/a/ws/2003/06/24/ws-trust.html?page=1 (accessed May 28, 2008).

McGovern J., Tyagi S., Stevens M. and Mathew S. 2003. *Java Web Services Architecture*. Morgan Kaufmann Publishers.

Microsoft Corporation 2002. Microsoft Corporation. http://samples.gotdotnet.com/quickstart/aspplus/doc/webservicesintro.aspx (accessed December 3, 2007).

Microsoft Patterns and Practices Developer Center 2005. Implementing Transport and Message Layer Security. http://msdn2.microsoft.com/en-us/library/aa480582.aspx (accessed April 7, 2008).

Montgomery D. 1997. *Introduction to Statistical Quality Control*. John Wiley & Dons, Inc.

Nakamura Y., Tatsubori M., Imamura T. and Ono K. 2005. *Model-Driven Security Based on a Web Services Security Architecture*. IBM Tokyo Research Laboratory.

Parducci B., Lockhart H. 2008. OASIS Extensible Access Control Markup Language (XACML), http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml (accessed September 20, 2007).

Raskin D. 2008. Identity Federation: The Inside Story (Literally). http://www.sun.com/emrkt/campaign_docs/idmgmt/newsletter/1107feature.html (accessed October 5, 2007).

RSA Laboratories 2007. RSA Laboratories, What are Message Authentication Codes. http://www.rsa.com/rsalabs/node.asp?id=2177 (accessed December 19, 2007).

Sehring H. 2006. Software Architectures. http://www.sts.tu-harburg.de (accessed December 18, 2007).

Stallings W. 2003. *Network Security Essentials Applications and Standards*. Prentice Hall.

Sun Microsystems Inc. 2006. The Soa Platform. http://www.sun.com/software/white papers/soa/soa_platform_guide.pdf (accessed February 18, 2008).

Sun Microsystems 2001. Sun Microsystems, Jini Network Technology. http://www.sun. com/software/jini/ (accessed November 27, 2007).

Thai T., Lam H. 2000. .*Net Framework Essentials*. O'Reilly, 3rd edition.

Thorsteinson P., Ganesh G. 2003. .*NET Security and Cryptography*. Prentice Hall.

W3Schools 2007. W3 Schools, Introduction to SOAP. http://www.w3schools.com/soap/ soap_intro.asp (accessed December 12, 2007).

Wahli U., Kjaer T., Robertson B., Satoh F., Schneider F., Szczeponik W. and Whyley C. 2006. *WebSphere Version 6: Web Services Handbook: Development and Deployment*. IBM Press.

APPENDIX A

WS-SECURITY EXAMPLE SOAP MESSAGES

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Header/> <soapenv:Body>
      <p821:getDayForecast xmlns:p821="http://bean.itso">
            <theDate>2004-11-25T15:00:00.000Z</theDate>
      </p821:getDayForecast></soapenv:Body>
      </soapenv:Envelope>
```

Figure A.1 SOAP message without WS-Security
(Source: Wahli, et al. 2006)

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header> <wsse:Security soapenv:mustUnderstand="1"
 xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
200401 - wsswssecurity- secext-1.0.xsd">
<wsse:UsernameToken><wsse:Username>David</wsse:Username>
<wsse:Password Type="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wssusername- token-
profile-1.0#PasswordText">divaD</wsse:Password>
            </wsse:UsernameToken>
         </wsse:Security>
      </soapenv:Header><soapenv:Body> <p821:getDayForecast
xmlns:p821="http://bean.itso"><theDate>2004-11-
25T15:00:00.000Z</theDate> </p821:getDayForecast>
      </soapenv:Body>
</soapenv:Envelope>
```

Figure A.2 SOAP Message with Authentication
(Source: Wahli, et al. 2006)

```
<saml:Assertion AssertionID="SecurityToken-07dc7c16-9a42-
4100-ad21-13013b975f3c"
MajorVersion="1" MinorVersion="1"
Issuer="http://localhost/SamlSecurityTokenService/SamlToken
Issuer.ashx"
IssueInstant="2005-11-29T22:36:03Z"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
<saml:Conditions NotBefore="2005-11-29T22:36:03Z"
NotOnOrAfter="2005-11-
30T02:36:03Z">
<saml:AudienceRestrictionCondition>
<saml:Audience>
http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anony
mous
</saml:Audience>
</saml:AudienceRestrictionCondition>
</saml:Conditions>
<saml:AuthenticationStatement
AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:passwo
rd"
AuthenticationInstant="2005-11-29T22:36:03Z">
<saml:Subject>
<saml:NameIdentifier
Format="http://schemas.xmlsoap.org/ws/2004/01/Federation/us
ername">RDALAPTOP02\wseuser</saml:NameIdentifier>
<saml:SubjectConfirmation>
<saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:hol
der-ofkey</
saml:ConfirmationMethod>
<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#rsa-1_5" />
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<wsse:SecurityTokenReference>
<wsse:KeyIdentifier ValueType="http://docs.oasisopen.
org/wss/oasis-wss-soap-message-security-1.1#ThumbprintSHA1"
EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-soapmessage-
security-
1.0#Base64Binary">aAI1zTqHbhsUN6j2HsIefWcHODs=</wsse:KeyIde
ntifier>
</wsse:SecurityTokenReference>
</KeyInfo>
```

Figure A.3 SOAP Message with Integrity
(Source: Wahli, et al. 2006)

(Cont. on next page)

```
<xenc:CipherData>
<xenc:CipherValue>cSrRHLh8DWeELh5Naf34z515OIs0sw6hp4/zUIOnb
OFNj1lxQdXCTi7z3aaLu4Xi
ws8vF3YdzD9LD/bQ1+QzzI7qcR4eDLNnxjZU87DkCBxI4ygqyB+Mx4J2lKL
Yl+rxI
OVOVcjbd64/YngQu5
AgZKBxNZv7GIcla0d3Ikebyr4=</xenc:CipherValue>
</xenc:CipherData>
</xenc:EncryptedKey>
</ds:KeyInfo>
</saml:SubjectConfirmation>
</saml:Subject>
<saml:SubjectLocality IPAddress="192.168.0.10"
DNSAddress="CLIENTHOST" />
</saml:AuthenticationStatement>
<saml:AttributeStatement>
<saml:Subject>
<saml:NameIdentifier
Format="http://schemas.xmlsoap.org/ws/2004/01/Federation/us
ername">RDALAPTOP02\wseuser</saml:NameIdentifier>
<saml:SubjectConfirmation>
<saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.0:cm:hol
der-ofkey</saml:ConfirmationMethod>
<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#rsa-1_5" />
<KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
<wsse:SecurityTokenReference>
<wsse:KeyIdentifier ValueType="http://docs.oasisopen.
org/wss/oasis-wss-soap-message-security-1.1#ThumbprintSHA1"
EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-soapmessage-
security-
1.0#Base64Binary">aAI1zTqHbhsUN6j2HsIefWcHODs=</wsse:KeyIde
ntifier>
</wsse:SecurityTokenReference>
</KeyInfo>
<xenc:CipherData>
<xenc:CipherValue>cSrRHLh8DWeELh5Naf34z515OIs0sw6hp4/zUIOnb
OFNj1lxQdXCTi7z3aaLu4Xi
ws8vF3YdzD9LD/bQ1+QzzI7qcR4eDLNnxjZU87DkCBxI4ygqyB+Mx4J2lKL
Yl+rxIOVOVcjbd64/YngQu5
AgZKBxNZv7GIcla0d3Ikebyr4=</xenc:CipherValue>
```

Figure A.3 (cont.) SOAP Message with Integrity
(Source: Wahli, et al. 2006)

(Cont. on next page)

```
</xenc:CipherData>
</xenc:EncryptedKey>
</ds:KeyInfo>
</saml:SubjectConfirmation>
</saml:Subject>
<saml:Attribute AttributeName="group"
AttributeNamespace="http://schemas.xmlsoap.org/ws/2004/01/F
ederation/group">
<saml:AttributeValue>BUILTIN\Users</saml:AttributeValue>
</saml:Attribute>
</saml:AttributeStatement>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-excc14n#"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#" />

<SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<Reference URI="#SecurityToken-07dc7c16-9a42-4100-ad21-
13013b975f3c">
<Transforms>
<Transform
Algorithm="http://www.w3.org/2000/09/xmldsig#envelopedsigna
ture"
/>
<Transform Algorithm="http://www.w3.org/2001/10/xml-exc-
c14n#" />
</Transforms>
<DigestMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<DigestValue>k2PBIDmwJLlQIh/GA4bVPgk1544=</DigestValue>
</Reference>
</SignedInfo>
<SignatureValue>CZPQJvc2vpjKyio6EEJTQShHGlJff1hZubOWSDCbRrb
SFkK53fOp9Ugulfel6vvs9y
ZfFJB2ieRAPK3ywUrRWTKvO7vjXP9HRGgrvqGC2PpQNAEYn7ciBkLM+VoJV
5vWfIopVevEYvnxFMFZlJTl
LSOr0n+GWYaZuUYFJECfHaE=</SignatureValue>
<KeyInfo>
<wsse:SecurityTokenReference>
<wsse:KeyIdentifier ValueType="http://docs.oasis-
open.org/wss/oasiswss-
soap-message-security-1.1#ThumbprintSHA1"
EncodingType="http://docs.oasisopen.
```

Figure A.3 (cont.) SOAP Message with Integrity
(Source: Wahli, et al. 2006)

(Cont. on next page)

```
org/wss/2004/01/oasis-200401-wss-soap-message-security-
1.0#Base64Binary">vLfjdZyxqw+ztcYkVVQX8wyXp5o=</wsse:KeyIde
ntifier>
</wsse:SecurityTokenReference>
</KeyInfo>
</Signature>
</saml:Assertion>
```

Figure A.3 (cont.) SOAP Message with Integrity
(Source: Wahli, et al. 2006)

```
<soapenv:Envelope
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <soapenv:Header>
            <wsu:Timestamp
                  xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-ws
                  security-utility-1.0.xsd">
                  <wsu:Created>2004-11-
26T09:34:50.838Z</wsu:Created>
             </wsu:Timestamp>
             <wsse:Security soapenv:mustUnderstand="1"
                  xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-w
                  ssecurity-secext-1.0.xsd">
                    <EncryptedKey
xmlns="http://www.w3.org/2001/04/xmlenc#">

 <EncryptionMethod

Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
                                    <ds:KeyInfo
xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

<wsse:SecurityTokenReference>
                                          <wsse:KeyIdentifier

ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
                                       200401-wss-x509-token-
```

Figure A.4 SOAP Message with Confidentiality
(Source: Wahli, et al. 2006)

(Cont. on next page)

```
profile-1.0#X509v3SubjectKeyIdentifier">
Vniy7MUOXBumPoH1MNbDpiIWOPA=
</wsse:KeyIdentifier>
</wsse:SecurityTokenReference>
                                    </ds:KeyInfo>
<CipherData>
                                   <CipherValue>
O+2mTsRjU1iNlwANv1kGdzpkRV1GQc5epAT3p5Eg5UNAJ3H3YAX5VrdgMQm
j1
wzdSZLDEzBtcHPJq3c8c0AgmAy9EVdcgXIn/ZeV+80jMDn/HN2HfodYjURt
IYBg48

     0SSkot0fy+YpBSXNR/MTfs1HT2H/Mjw/CyIbomWdQZHmE=
</CipherValue>
</CipherData>
<ReferenceList>

<DataReferenceURI="#wssecurity_encryption_id_68669508378406
88804"/>
                             </ReferenceList>
                   </EncryptedKey>
            </wsse:Security>
      </soapenv:Header>
      <soapenv:Body>
            <EncryptedData

Id="wssecurity_encryption_id_68669508378406888804"

Type="http://www.w3.org/2001/04/xmlenc#Content"

xmlns="http://www.w3.org/2001/04/xmlenc#">
                  <EncryptionMethod

Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-
cbc"/>
                  <CipherData>
                        <CipherValue>
OvLekOlbuZhFBllBNL4Kos195YHwYw0kSbMxkbI2pk7nl17g0prPS2Ba2hy
rXHABGQVmosWpgqt+zijCPHUQCMwmm3qgFraK11DPMmwP94HvgxlgBmPw1U
nt+WM4aKLNrHDnwwcQX5RO7KT+fhFp4wxFEABwfHqzvTGNK3xRwJE=
                         </CipherValue>
                  </CipherData>
            </EncryptedData>
      </soapenv:Body>
     </soapenv:Envelope>
```

Figure A.4 (cont.) SOAP Message with Confidentiality
(Source: Wahli, et al. 2006)

APPENDIX B

CLASS DIAGRAMS

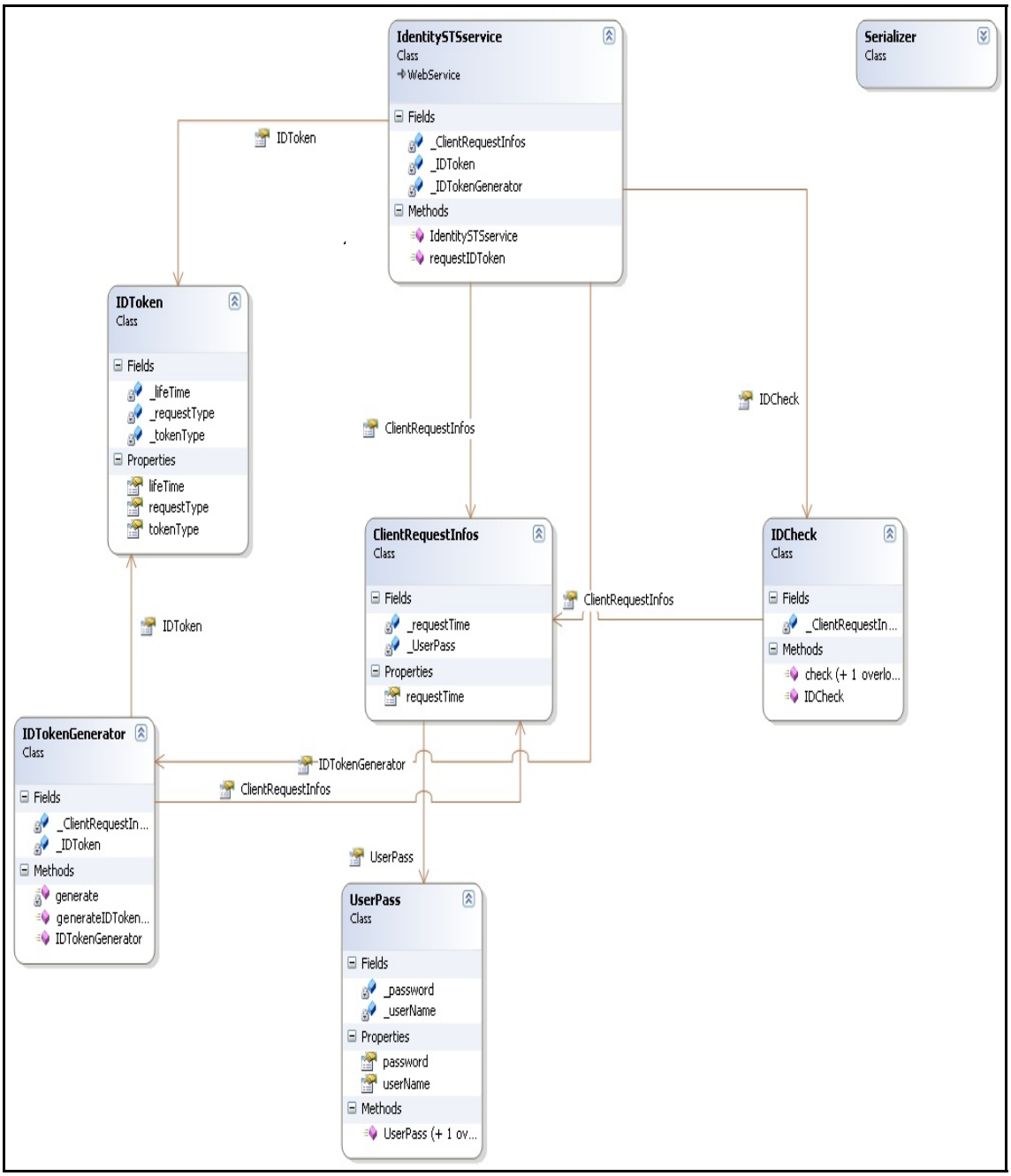

Figure B.1. Class Diagram of Identity STS

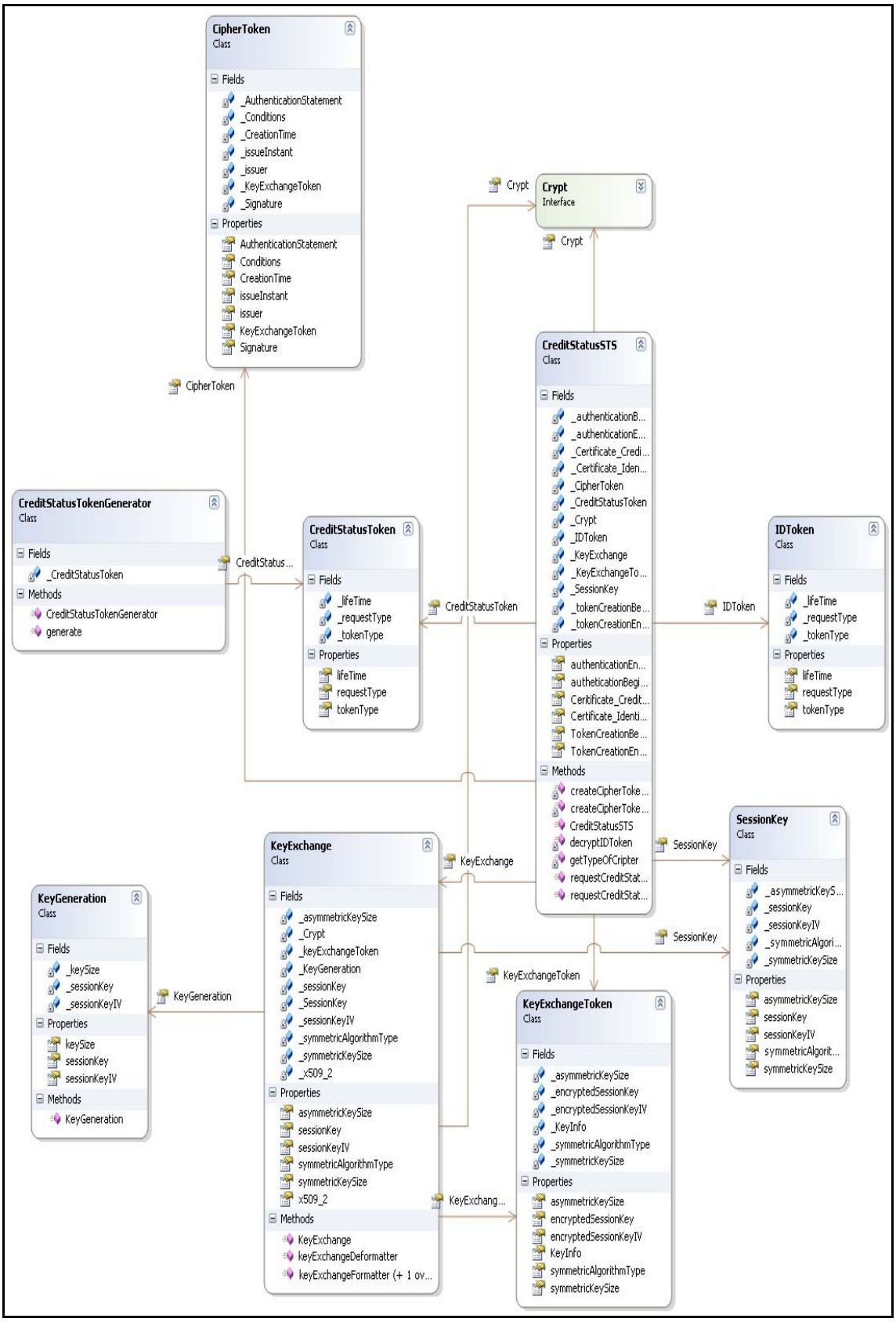

Figure B.2. Class Diagram of Credit Status STS

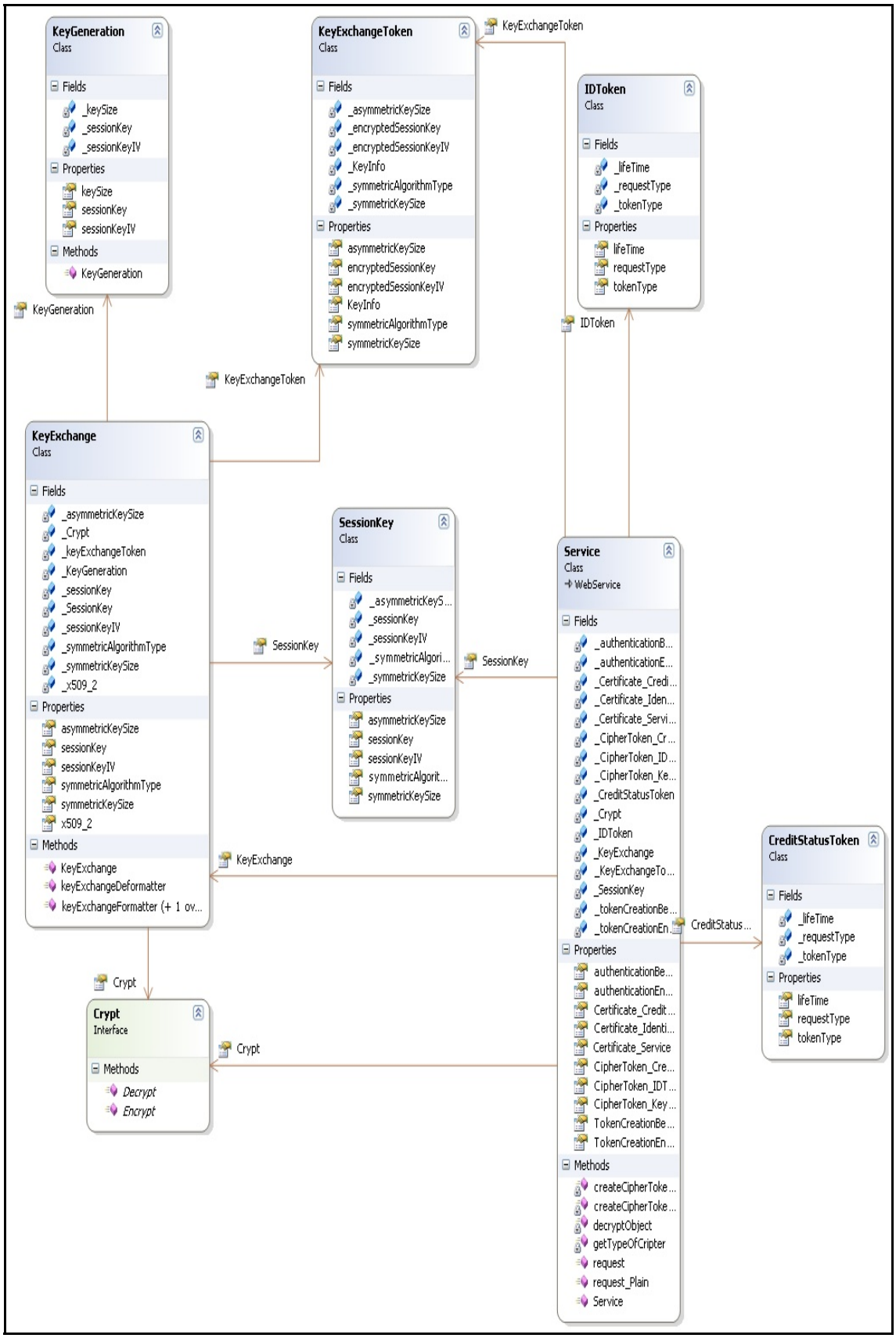

Figure B.3. Class Diagram of Service STS

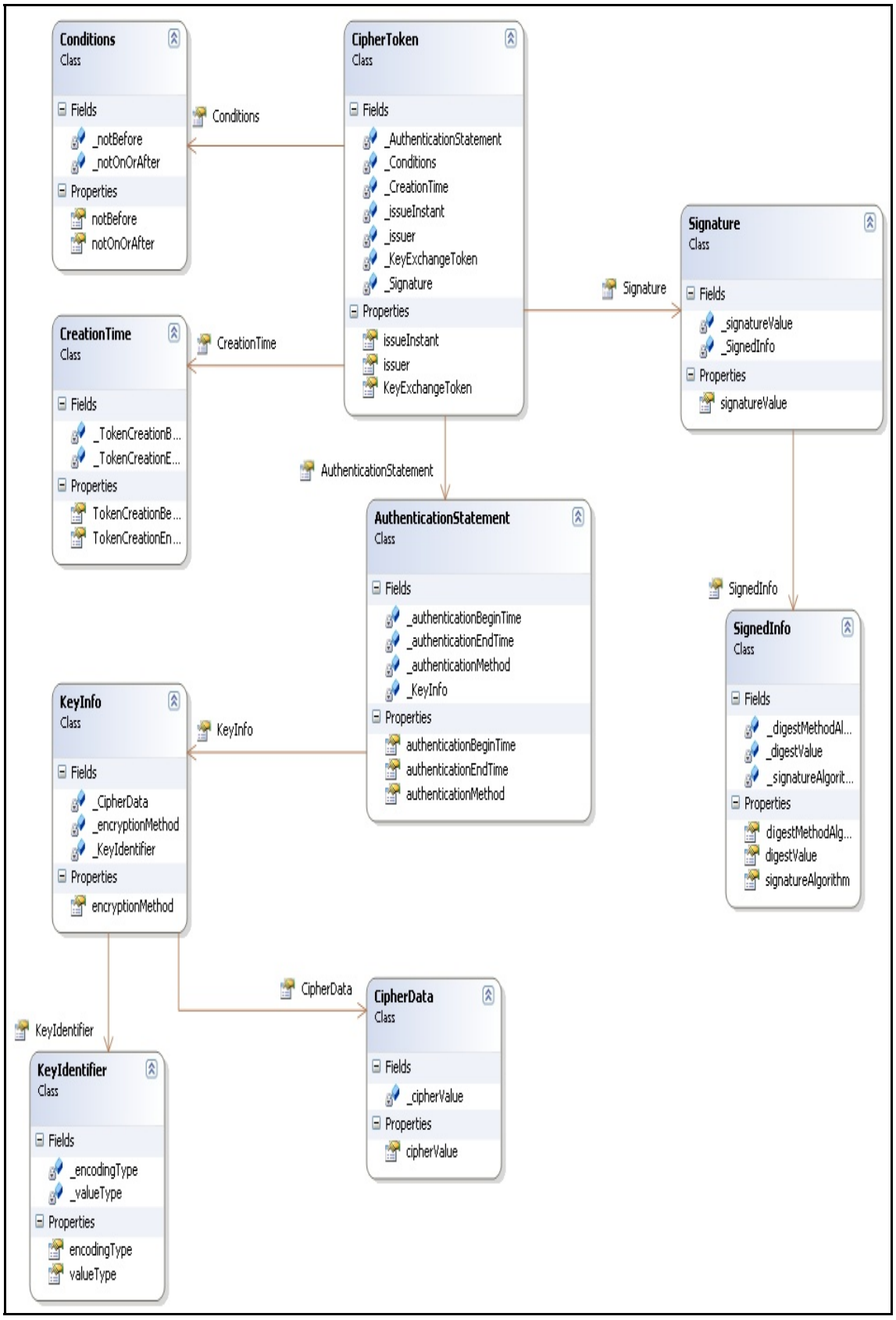

Figure B.4. Cipher Token Class Diagram

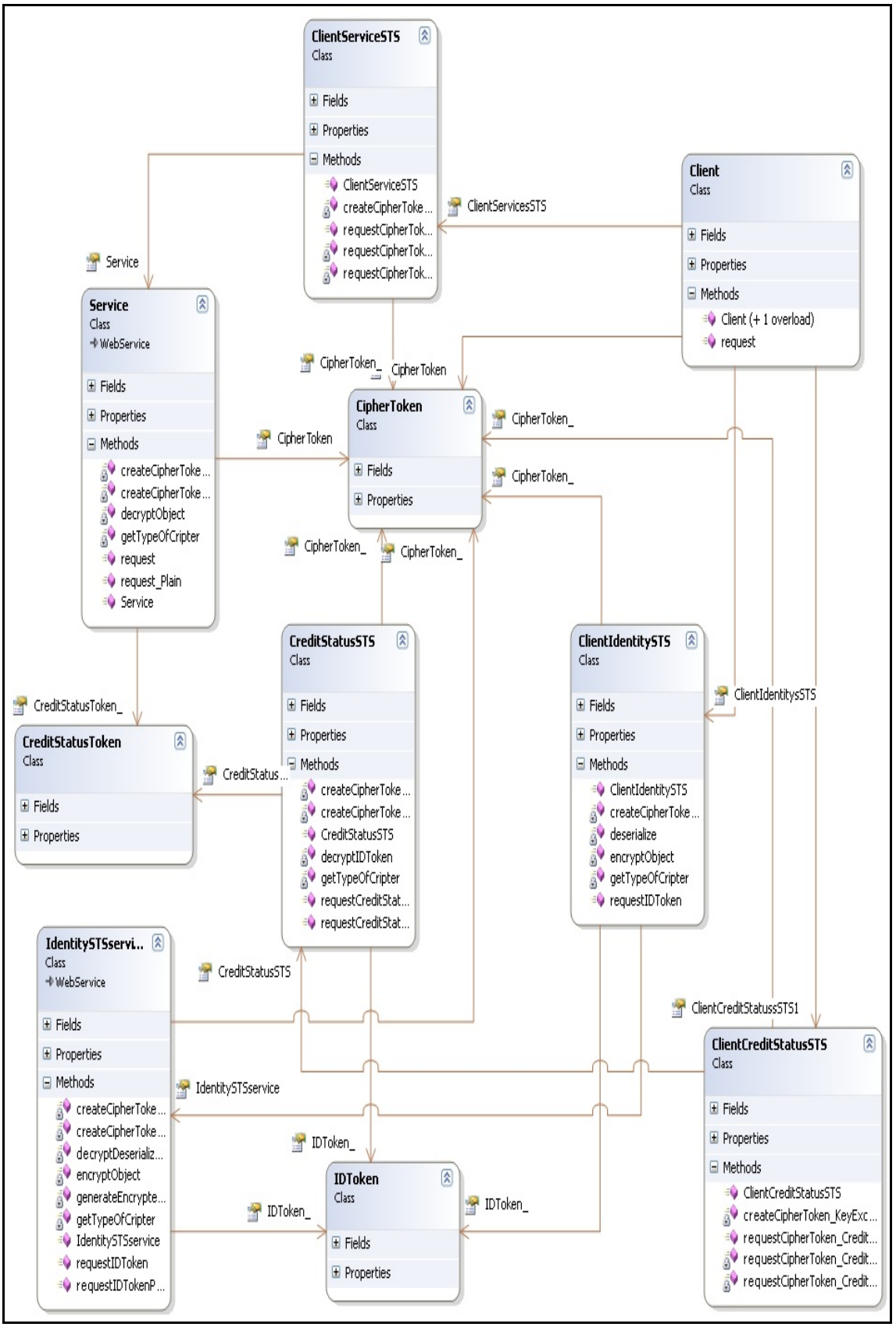

Figure B.5. Domain Model

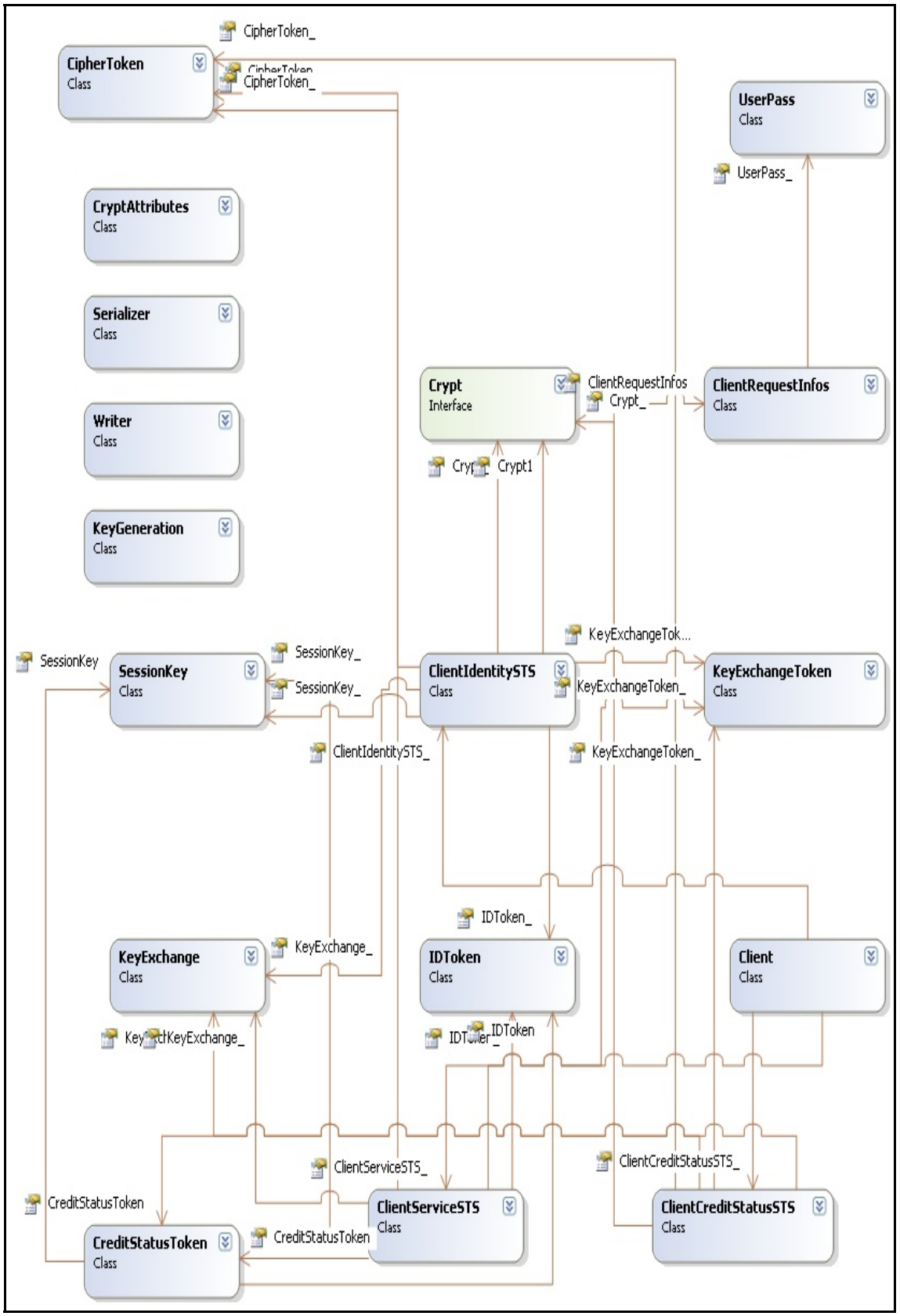

Figure B.6. Client Class Diagram

APPENDIX C

WEB SERVICES SOAP MESSAGE EXAMPLES

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <requestIDToken xmlns="http://tempuri.org/">
      <serializedCipherToken_KeyExchangeToken>
<CipherToken   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Conditions>
    <notBefore>2008-05-25T21:41:52.546875+03:00</notBefore>
<notOnOrAfter>2008-05-
25T21:56:52.546875+03:00</notOnOrAfter>
  </Conditions>
  <AuthenticationStatement>
    <KeyInfo>
      <CipherData>
<cipherValue>PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLT
E2Ij8+DQo8S2V5RXhjaGFuZ2VUb2tlbiB4bWxuczp4c2k9Imh0dHA6Ly93d
3cudzMub3JnLzIwMDEvWE1MU2NoZW1hLWluc3RhbmNlIiB4bWxuczp4c2Q9
Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1hIj4NCiAgPGVuY3J
5cHRlZFNlc3Npb25LZXk+cldUbllBaGZld0pQOHZPaU1EUExuOU1iZElPMU
```

Figure C.1. SOAP Message Example For Identity Token Request From Client To Identity Token Web Service

(Cont. on next page)

```
Fvc1FUT3A3NTk1aFd5NkpoZlVrUTJNZllnV2pQc0xFajh0aExEUFJBZDJXb
WFKbDVRa2hxa0ZGS29POVdOMmFudU1neHdURE5LUnU2RGprMjhuMnpmTmZJ
aXdLazJQVDVEc0RaWUdpTHlBZkRRRFNFTUh0QlUxREFlTlpzSXpiaEtqaWN
tOWtid2xPSC9JPTwvZW5jcnlwdGVkU2Vzc2lvbktleT4NCiAgPHN5bW1ldH
JpY0FsZ29yaXRobVR5cGU+QUVTPC9zeW1tZXRyaWNBbGdvcml0aG1UeXBlP
g0KICA8ZW5jcnlwdGVkU2Vzc2lvbktleUlWPmd5OXR4b3grVVFSUU1QVFFX
TTRTZ3lqUU90WDFUUEQzS0g5d3Rua1J4WGw1c0JLUUtUWFRvL2FXcWMzbzV
ENXgrUHBmVDlyQ1dFSGprK09KdWY3b2xFalNubE9LSkNFUUtuZlFIUktBV1
U4YmhRKyttckl4eU10bjFBeC8xOS9IeGgzYTR4L2pGekdBMTltRVU3RTZJU
2NQQ2s4bFRqRHdabjNETGVDUmx5WT08L2VuY3J5cHRlZFNlc3Npb25LZXlJ
Vj4NCiAgPHN5bW1ldHJpY0tleVNpemU+NjQ8L3N5bW1ldHJpY0tleVNpemU
+DQogIDxhc3ltbWV0cmljS2V5U2l6ZT4xMDI0PC9hc3ltbWV0cmljS2V5U2
l6ZT4NCiAgPEtleUluZm8+DQogICAgPGVuY3J5cHRpb25NZXRob2Q+UlNBL
UFFUzwvZW5jcnlwdGlvbk1ldGhvZD4NCiAgPC9LZXlJbmZvPg0KPC9LZXlF
eGNoYW5nZVRva2VuPg==</cipherValue></CipherData>
      <encryptionMethod>RSA-AES</encryptionMethod>
    </KeyInfo>
    <authenticationEndTime>2008-05-
25T21:41:52.546875+03:00</authenticationEndTime>
<authenticationMethod>UserName_Password</authenticationMeth
od>
    <authenticationBeginTime>0001-01-
01T00:00:00</authenticationBeginTime>
  </AuthenticationStatement>
  <issueInstant>2008-05-
25T21:41:52.5625+03:00</issueInstant>
  <issuer>Client</issuer>
</CipherToken>
</serializedCipherToken_KeyExchangeToken>
```

Figure C.1. (cont.) SOAP Message Example For Identity Token Request From Client To Identity Token Web Service

(Cont. on next page)

```
 <serializedCipherToken_CRI>
<CipherToken   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<Conditions>
    <notBefore>2008-05-25T21:41:52.953125+03:00</notBefore>
    <notOnOrAfter>2008-05-
25T21:56:52.953125+03:00</notOnOrAfter>
  </Conditions>
  <AuthenticationStatement>
    <KeyInfo>
      <CipherData>
<cipherValue>y2Y67tnzZzTXNGvZQsU2EI4sKbc32/cJrZkf/p2/U0AWls
rpRhsv1e8tlWrVFlLdtEX5dKgcRxdH8mjGtQbmbtNCc2+p5pArEn26L4oCp
P5EyaX5nQFzzoMBUQgQ5BgMdHOnR1Py/XrKTo8k6Nf817cM8CmfhAUrj7Sp
F7eWpE5UXHYQxmQTpcQTRFZRP5U2lrN6eWi5Ye+tiQVpTsJt+D3K6h0KlSx
/rIhcAyGOPRO874KTvTQQJI3r12Rd2EOIIuclyS/zAnhjMZbrgbl6mFCtDp
5oLqGxy2HtTKl1a2crO+4YRQKCm/Ke+0aI9tU+bf10gQbMkqOBJ22ce66sl
ZYoVEhmfYyAL6o9GGxdS92z171EhvMQZG8jfznUixoITDDrRPob9rqlsHb3
3GattFkDLeOemH7spaQsBKLsKlrPcO5TgP2rinWQBphjaZ0D
</cipherValue>

</CipherData>
      <encryptionMethod>RSA-AES</encryptionMethod>
    </KeyInfo>
    <authenticationEndTime>2008-05-
25T21:41:52.953125+03:00</authenticationEndTime>
<authenticationMethod>UserName_Password</authenticationMeth
od>
```

Figure C.1. (cont.) SOAP Message Example For Identity Token Request From Client To Identity Token Web Service

(Cont. on next page)

```
    <authenticationBeginTime>0001-01-
01T00:00:00</authenticationBeginTime>
  </AuthenticationStatement>
  <issueInstant>2008-05-
25T21:41:52.953125+03:00</issueInstant>
  <issuer>Client</issuer></CipherToken>
</serializedCipherToken_CRI>
    </requestIDToken>
  </soap:Body>
</soap:Envelope>
```

Figure C.1. (cont.) SOAP Message Example For Identity Token Request From Client To Identity Token Web Service

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <requestIDTokenResponse xmlns="http://tempuri.org/">
      <requestIDTokenResult>
<CipherToken   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Conditions>
    <notBefore>2008-05-25T21:41:54.3125+03:00</notBefore>
    <notOnOrAfter>2008-05-
25T21:56:54.3125+03:00</notOnOrAfter>
  </Conditions>
  <AuthenticationStatement>
```

Figure C.2. SOAP Message Example For Identity Token Response From Identity STS to Client

(Cont. on next page)

```
<KeyInfo>
      <CipherData>

<cipherValue>y2Y67tnzZzTXNGvZQsU2EI4sKbc32/cJrZkf/p2/U0DyUJ
2REyQvKZZn5gz0caNdmXT3nTcOeVKxgJNY5JAql3iHq/XJo1D6h99QrRpTh
Ll7tz7LdGajmAKZ+g1hQef+zMDx9gkzh8+qSkAE84DmTtScP0cRTwUnstJT
+dzEdxlpMffSNvjXLOszdfXyIj0qi1HDEo/xCWQxUDljz2iYbqQHIS2zdeG
zCgEilu3t74lTzeh2gFM1LOobwbqLyGo2801EiCftqojeokMRHnXKFAzXHk
u/lz+5hN1CHl2Cdp4Q4O9jUaVFU5AHeRLGv7+KTLCA+noIMdz1Xc6cpU36X
Pr+kcsWYsztRiRFLdyOTyVwC+GRQR83oIMhaf+Y4wnI
</cipherValue>
      </CipherData>
      <encryptionMethod>RSA-AES</encryptionMethod>
    </KeyInfo>
    <authenticationEndTime>2008-05-
25T21:41:54.3125+03:00</authenticationEndTime>
<authenticationMethod>UsernamePassword</authenticationMetho
d>
    <authenticationBeginTime>2008-05-
25T21:41:53.640625+03:00</authenticationBeginTime>
  </AuthenticationStatement>
  <Signature>
    <SignedInfo>
      <signatureAlgorithm>SHA1</signatureAlgorithm>
    </SignedInfo>

<signatureValue>GD6xPMaN2W9+jM3wTvCbMjP3dqU+gh5ldUxOS6vdERt
X6VKuCE4szcf3Qom5Mv9JL5GPm6+0+badyrKb2iDB4AqQMj8lVyOB9buPcf
0SoeQM4mhbINIIGiTNmvdRfghpqrKhDAUVZHZ7z/xPvXMvSzUtYDs86+xwJ
```

Figure C.2. (cont.) SOAP Message Example For Identity Token Response From Identity STS to Client

(Cont. on next page)

```
iEudI9zDp0=</signatureValue>
  </Signature><issueInstant>2008-05-
25T21:41:54.46875+03:00</issueInstant>
  <issuer>Client</issuer>
  <KeyExchangeToken>
<encryptedSessionKey>rWTnYAhfewJP8vOiMDPLn9MbdIO1AosQTOp759
5hWy6JhfUkQ2MfYgWjPsLEj8thLDPRAd2WmaJl5QkhqkFFKoO9WN2anuMgx
wTDNKRu6Djk28n2zfNfIiwKk2PT5DsDZYGiLyAfDQDSEMHtBU1DAeNZsIzb
hKjicm9kbwlOH/I=</encryptedSessionKey>
    <symmetricAlgorithmType>AES</symmetricAlgorithmType>
<encryptedSessionKeyIV>gy9txox+UQRQMPTQWM4SgyjQOtX1TPD3KH9w
tnkRxXl5sBKQKTXTo/aWqc3o5D5x+PpfT9rCWEHjk+OJuf7olEjSnlOKJCE
QKnfQHRKAWU8bhQ++mrIxyMtn1Ax/19/Hxh3a4x/jFzGA19mEU7E6IScPCk
8lTjDwZn3DLeCRlyY=</encryptedSessionKeyIV>
    <symmetricKeySize>64</symmetricKeySize>
    <asymmetricKeySize>1024</asymmetricKeySize>
    <KeyInfo>
      <encryptionMethod>RSA-AES</encryptionMethod>
    </KeyInfo>
  </KeyExchangeToken>
  <CreationTime>
    <TokenCreationBeginTime>2008-05-
25T21:41:54.3125+03:00</TokenCreationBeginTime>
    <TokenCreationEndTime>2008-05-
25T21:41:54.46875+03:00</TokenCreationEndTime>
  </CreationTime>
</CipherToken>
</requestIDTokenResult></requestIDTokenResponse>
  </soap:Body>
</soap:Envelope>
```

Figure C.2. (cont.) SOAP Message Example For Identity Token Response From Identity STS to Client

```
<?xml version="1.0" encoding="utf-16"?>
<ClientRequestInfos
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <UserPass>
    <userName>user0</userName>
    <password>pass0</password>
  </UserPass>
  <requestTime>0001-01-01T00:00:00</requestTime>
</ClientRequestInfos>
```

Figure C.3. Client Request Information Open Message

```
<?xml version="1.0" encoding="utf-16"?>
<IDToken       xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <lifeTime>2008-05-26T21:41:54.3125+03:00</lifeTime>
  <requestType>pass0</requestType>
  <tokenType>user0</tokenType>
</IDToken>
```

Figure C.4. IDToken Open Message

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <requestCreditStatus xmlns="http://tempuri.org/">
```

Figure C.5. SOAP Message Example For Credit Status Token Request From Client To CreditStatusSTS Web Service

(Cont. on next page)

<serializedCipherToken_KeyExchangeToken>

```
<CipherToken   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Conditions>
    <notBefore>2008-05-25T21:41:55.109375+03:00</notBefore>
    <notOnOrAfter>2008-05-
25T21:56:55.109375+03:00</notOnOrAfter>
  </Conditions>
  <AuthenticationStatement>
    <KeyInfo>
      <CipherData>

<cipherValue>PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLT
E2Ij8+DQo8S2V5RXhjaGFuZ2VUb2tlbiB4bWxuczp4c2k9Imh0dHA6Ly93d
3cudzMub3JnLzIwMDEvWE1MU2NoZW1hLWluc3RhbmNlIiB4bWxuczp4c2Q9
Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1hIj4NCiAgPGVuY3J
5cHRlZFNlc3Npb25LZXk+RTF1OUNSbTZJcDlEZHZ1V3BXaUx0ZTdTeEhZQ0
FUMVUxY2tkb0J5OE9IbHBjUktQU25pdWZsSkV3MTcrazYydGhzZUZOV0pPU
jVCRUdaSXB3cy8zeUhuUldnT3ZtdUdWUGdJZHVkVHBrbE16Z2xXTGJ4ZDJs
Q1JqMy9rRkwvN1R3M0p2WTRtS3ByNEtWeVJYWnZsMExiTnRDYmJ4MGtzSEd
VL3VrVzVmUWxBPTwvZW5jcnlwdGVkU2Vzc2lvbktleT4NCiAgPHN5bW1ldH
JpY0FsZ29yaXRobVR5cGU+QUVTPC9zeW1tZXRyaWNBbGdvcml0aG1UeXBlP
g0KICA8ZW5jcnlwdGVkU2Vzc2lvbktleUlWPk1wcEZRbVg3aGRzZ2V2dXlv
czVXYUxDSFdZN3Q2THpuY2dNSmF6M2xZenl3b1hCeUdBSy9QaXRac1dpV1h
lc044VCthUTJNSEZWVUN2MFZEUTJBZ0dFWEtVK1RpNFB2SndQK2FaVGsySk
U5emYyNk1paUt4MWgxcWNmUThQelJpSjM4QmhIcktud0lUZDc2VS9pV1ZNd
0Q0MzN6c1BTRkI1cER6M3R4Q21TRT08L2VuY3J5cHRlZFNlc3Npb25LZXlJ
Vj4NCiAgPHN5bW1ldHJpY0tleVNpemU+NjQ8L3N5bW1ldHJpY0tleVNpemU
```

Figure C.5. (cont.) SOAP Message Example For Credit Status Token Request From Client To CreditStatusSTS Web Service

(Cont. on next page)

```
+DQogIDxhc3ltbWV0cmljS2V5U2l6ZT4xMDI0PC9hc3ltbWV0cmljS2V5U2
l6ZT4NCiAgPEtleUluZm8+DQogICAgPGVuY3J5cHRpb25NZXRob2Q+UlNBL
UFFUzwvZW5jcnlwdGlvbk1ldGhvZD4NCiAgPC9LZXlJbmZvPg0KPC9LZXlF
eGNoYW5nZVRva2VuPg==</cipherValue>
      </CipherData>
      <encryptionMethod>RSA-TripleDES</encryptionMethod>
    </KeyInfo>
    <authenticationEndTime>2008-05-
25T21:41:55.109375+03:00</authenticationEndTime>
<authenticationMethod>UserName_Password</authenticationMeth
od>
    <authenticationBeginTime>0001-01-
01T00:00:00</authenticationBeginTime>
  </AuthenticationStatement>
  <issueInstant>2008-05-
25T21:41:55.109375+03:00</issueInstant>
  <issuer>Client</issuer> </CipherToken>
</serializedCipherToken_KeyExchangeToken>
      <serializedCipherToken_IDtoken>
      <CipherToken
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <Conditions>
         <notBefore>2008-05-
25T21:41:54.3125+03:00</notBefore>
         <notOnOrAfter>2008-05-
25T21:56:54.3125+03:00</notOnOrAfter>
       </Conditions>
       <AuthenticationStatement>
```

Figure C.5. (cont.) SOAP Message Example For Credit Status Token Request From Client To CreditStatusSTS Web Service

(Cont. on next page)

```
     <KeyInfo>
     <CipherData>
<cipherValue>y2Y67tnzZzTXNGvZQsU2EI4sKbc32/cJrZkf/p2/U0DyUJ
2REyQvKZZn5gz0caNdmXT3nTcOeVKxgJNY5JAql3iHq/XJo1D6h99QrRpTh
Ll7tz7LdGajmAKZ+g1hQef+zMDx9gkzh8+qSkAE84DmTtScP0cRTwUnstJT
+dzEdxlpMffSNvjXLOszdfXyIj0qi1HDEo/xCWQxUDljz2iYbqQHIS2zdeG
zCgEilu3t74lTzeh2gFM1LOobwbqLyGo2801EiCftqojeokMRHnXKFAzXHk
u/lz+5hN1CHl2Cdp4Q4O9jUaVFU5AHeRLGv7+KTLCA+noIMdz1Xc6cpU36X
Pr+kcsWYsztRiRFLdyOTyVwC+GRQR83oIMhaf+Y4wnI</cipherValue>
          </CipherData>
          <encryptionMethod>RSA-AES</encryptionMethod>
        </KeyInfo>
        <authenticationEndTime>2008-05-
25T21:41:54.3125+03:00</authenticationEndTime>
<authenticationMethod>UsernamePassword</authenticationMetho
d>
        <authenticationBeginTime>2008-05-
25T21:41:53.640625+03:00</authenticationBeginTime>
      </AuthenticationStatement>
      <Signature>
        <SignedInfo>
          <signatureAlgorithm>SHA1</signatureAlgorithm>
        </SignedInfo>
<signatureValue>GD6xPMaN2W9+jM3wTvCbMjP3dqU+gh5ldUxOS6vdERt
X6VKuCE4szcf3Qom5Mv9JL5GPm6+0+badyrKb2iDB4AqQMj8lVyOB9buPcf
0SoeQM4mhbINIIGiTNmvdRfghpqrKhDAUVZHZ7z/xPvXMvSzUtYDs86+xwJ
iEudI9zDp0=</signatureValue>
     </Signature>
     <issueInstant>2008-05-
```

Figure C.5. (cont.) SOAP Message Example For Credit Status Token Request From Client To CreditStatusSTS Web Service

(Cont. on next page)

```
     25T21:41:54.46875+03:00</issueInstant>

       <issuer>Client</issuer>
       <KeyExchangeToken>
<encryptedSessionKey>rWTnYAhfewJP8vOiMDPLn9MbdIO1AosQTOp759
5hWy6JhfUkQ2MfYgWjPsLEj8thLDPRAd2WmaJl5QkhqkFFKoO9WN2anuMgx
wTDNKRu6Djk28n2zfNfIiwKk2PT5DsDZYGiLyAfDQDSEMHtBU1DAeNZsIzb
hKjicm9kbwlOH/I=</encryptedSessionKey>
<symmetricAlgorithmType>AES</symmetricAlgorithmType>
<encryptedSessionKeyIV>gy9txox+UQRQMPTQWM4SgyjQOtX1TPD3KH9w
tnkRxXl5sBKQKTXTo/aWqc3o5D5x+PpfT9rCWEHjk+OJuf7olEjSnlOKJCE
QKnfQHRKAWU8bhQ++mrIxyMtn1Ax/19/Hxh3a4x/jFzGA19mEU7E6IScPCk
8lTjDwZn3DLeCRlyY=</encryptedSessionKeyIV>
         <symmetricKeySize>64</symmetricKeySize>
         <asymmetricKeySize>1024</asymmetricKeySize>
         <KeyInfo>
           <encryptionMethod>RSA-AES</encryptionMethod>
         </KeyInfo>
       </KeyExchangeToken>
       <CreationTime>
         <TokenCreationBeginTime>2008-05-
25T21:41:54.3125+03:00</TokenCreationBeginTime>
         <TokenCreationEndTime>2008-05-
25T21:41:54.46875+03:00</TokenCreationEndTime>
       </CreationTime>
     </CipherToken>
     </serializedCipherToken_IDtoken>
    </requestCreditStatus>
  </soap:Body>
</soap:Envelope>
```

Figure C.5. (cont.) SOAP Message Example For Credit Status Token Request From Client To CreditStatusSTS Web Service

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
<requestCreditStatusResponsexmlns="http://tempuri.org/">
      <requestCreditStatusResult>
      CipherToken
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <Conditions>
         <notBefore>2008-05-
25T21:41:56.28125+03:00</notBefore>
         <notOnOrAfter>2008-05-
25T21:56:56.28125+03:00</notOnOrAfter>
       </Conditions>
       <AuthenticationStatement>
         <KeyInfo>
           <CipherData>
<cipherValue>y2Y67tnzZzTXNGvZQsU2EI4sKbc32/cJrZkf/p2/U0AMpr
2OeH50HcgQzZielTODCjGL8M+/y2iaZXFnxNSR7bLiD3BteytXP0LujYBZk
Pkdfr9JqtbSRjB1vs83uGF2kAVZ6A82ByEQTD4hr7te0l/QGJHP0CkqBId1
aLWmb5Knm5N5EWc2aWziWiEV9M7jIikZh9SUsGJH6qvjgmXOAxUYLH5qlfY
qJReFioKVi6K0cv9o6q0o6I9yqWbPo+7YWxQyXahNEO4KcMFH5H41UuJhc1
42rJG1PiDTFvfB+lzvy0jIGmUQEc06ld6f4DmKuTAHKKEthCbMCwUiWH3MF
KRBR5sNd7ymhR0qJGZAahO/bfWtCNyrO5hPTJOWlnULBpJPVIDkifJOhLVu
1agyGKrs/z3HGfmG17rTEleU6vfjaQ+y7EOcSGmTsHvhJ0C+</cipherVal
ue>
</CipherData>
```

Figure C.6. SOAP Message Example For Credit Status Token Response From Credit Status Token Web Service To Client

(Cont. on next page)

```
        <encryptionMethod>RSA-AES</encryptionMethod>
          </KeyInfo>
          <authenticationEndTime>2008-05-
25T21:41:55.984375+03:00</authenticationEndTime>
<authenticationMethod>SignatureAuthentication</authenticati
onMethod><authenticationBeginTime>2008-05-
25T21:41:55.453125+03:00</authenticationBeginTime>
        </AuthenticationStatement>
        <Signature>
          <SignedInfo>
            <signatureAlgorithm>SHA1</signatureAlgorithm>
          </SignedInfo>
<signatureValue>qt/oNbCHxYTIrLl/F/vuaE4DuL6LgCsBxsOGXZj4QXH
R/a7Q2xESaFD2ihBAs6c5yeh3YrtNYjZXasSorbJem5sRMIlnHRuGt7RizE
5F63Wd+XOf88mElilPIooBl0of15T/eDtpFxK5TJ26r5v47nTNInokv0kwK
vDJwAF6FVE=</signatureValue>
        </Signature> <issueInstant>2008-05-
      25T21:41:56.28125+03:00</issueInstant>
        <issuer>CreditStatusSTS</issuer>
        <CreationTime>
          <TokenCreationBeginTime>2008-05-
25T21:41:55.453125+03:00</TokenCreationBeginTime>
          <TokenCreationEndTime>2008-05-
25T21:41:56.28125+03:00</TokenCreationEndTime>
        </CreationTime>
      </CipherToken>
       </requestCreditStatusResult>
    </requestCreditStatusResponse>
  </soap:Body>
</soap:Envelope>
```

Figure C.6. (cont.) SOAP Message Example For Credit Status Token Response From Credit Status Token Web Service To Client

```
<?xml version="1.0" encoding="utf-16"?>
<CreditStatusToken
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <lifeTime>2008-05-26T21:41:56.125+03:00</lifeTime>
  <requestType>CreditStatusRequest</requestType>
  <tokenType>CreditStatusSTSToken</tokenType>
</CreditStatusToken>
```

Figure C.7. Credit Status Token Open Message

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <request xmlns="http://tempuri.org/">
      <serializedCipherToken_KeyExchangeToken>
      <CipherToken
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <Conditions>
         <notBefore>2008-05-
25T21:41:52.546875+03:00</notBefore>
         <notOnOrAfter>2008-05-
25T21:56:52.546875+03:00</notOnOrAfter>
       </Conditions>
       <AuthenticationStatement>
         <KeyInfo>
```

Figure C.8. SOAP Message Example For Service Request From Client to Service Web Service

(Cont. on next page)

```
     <CipherData>
<cipherValue>PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLT
E2Ij8+DQo8S2V5RXhjaGFuZ2VUb2tlbiB4bWxuczp4c2k9Imh0dHA6Ly93d
3cudzMub3JnLzIwMDEvWE1MU2NoZW1hLWluc3RhbmNlIiB4bWxuczp4c2Q9
Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1hIj4NCiAgPGVuY3J
5cHRlZFNlc3Npb25LZXk+cldUbllBaGZld0pQOHZPaU1EUExuOU1iZElPMU
Fvc1FUT3A3NTk1aFd5NkpoZlVrUTJNZllnV2pQc0xFajh0aExEUFJBZDJXb
WFKbDVRa2hxa0ZGS29POVdOMmFudU1neHdURE5LUnU2RGprMjhuMnpmTmZJ
aXdLazJQVDVEc0RaWUdpTHlBZkRRRFNFTUh0QlUxREFlTlpzSXpiaEtqaWN
tOWtid2xPSC9JPTwvZW5jcnlwdGVkU2Vzc2lvbktleT4NCiAgPHN5bW1ldH
JpY0FsZ29yaXRobVR5cGU+QUVTPC9zeW1tZXRyaWNBbGdvcml0aG1UeXBlP
g0KICA8ZW5jcnlwdGVkU2Vzc2lvbktleUlWPmd5OXR4b3grVVFSUU1QVFFX
TTRTZ3lqUU90WDFUUEQzS0g5d3Rua1J4WGw1c0JLUUtUWFRvL2FXcWMzbzV
ENXgrUHBmVDlyQ1dFSGprK09KdWY3b2xFalNubE9LSkNFUUtuZlFIUktBV1
U4YmhRKyttckl4eU10bjFBeC8xOS9IeGgzYTR4L2pGekdBMTltRVU3RTZJU
2NQQ2s4bFRqRHdabjNETGVDUmx5WT08L2VuY3J5cHRlZFNlc3Npb25LZXlJ
Vj4NCiAgPHN5bW1ldHJpY0tleVNpemU+NjQ8L3N5bW1ldHJpY0tleVNpemU
+DQogIDxhc3ltbWV0cmljS2V5U2l6ZT4xMDI0PC9hc3ltbWV0cmljS2V5U2
l6ZT4NCiAgPEtleUluZm8+DQogICAgPGVuY3J5cHRpb25NZXRob2Q+UlNBL
UFFUzwvZW5jcnlwdGlvbk1ldGhvZD4NCiAgPC9LZXlJbmZvPg0KPC9LZXlF
eGNoYW5nZVRva2VuPg==</cipherValue>
          </CipherData>
          <encryptionMethod>RSA-AES</encryptionMethod>
        </KeyInfo>
        <authenticationEndTime>2008-05-
25T21:41:52.546875+03:00</authenticationEndTime>
<authenticationMethod>UserName_Password</authenticationMeth
od>
        <authenticationBeginTime>0001-01-
```

Figure C.8. (cont.) SOAP Message Example For Service Request From Client to Service Web Service

(Cont. on next page)

```
     01T00:00:00</authenticationBeginTime>

       </AuthenticationStatement>
       <issueInstant>2008-05-
25T21:41:52.5625+03:00</issueInstant>
       <issuer>Client</issuer>
     </CipherToken>
    </serializedCipherToken_KeyExchangeToken>
   <serializedCipherToken_IDToken>
      <CipherToken
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
       <Conditions>
         <notBefore>2008-05-
25T21:41:54.3125+03:00</notBefore>
         <notOnOrAfter>2008-05-25T21:56:54.3125+03:00
     </notOnOrAfter>
       </Conditions>
       <AuthenticationStatement>
         <KeyInfo>
           <CipherData>
<cipherValue>y2Y67tnzZzTXNGvZQsU2EI4sKbc32/cJrZkf/p2/U0DyUJ
2REyQvKZZn5gz0caNdmXT3nTcOeVKxgJNY5JAql3iHq/XJo1D6h99QrRpTh
Ll7tz7LdGajmAKZ+g1hQef+zMDx9gkzh8+qSkAE84DmTtScP0cRTwUnstJT
+dzEdxlpMffSNvjXLOszdfXyIj0qi1HDEo/xCWQxUDljz2iYbqQHIS2zdeG
zCgEilu3t74lTzeh2gFM1LOobwbqLyGo2801EiCftqojeokMRHnXKFAzXHk
u/lz+5hN1CHl2Cdp4Q4O9jUaVFU5AHeRLGv7+KTLCA+noIMdz1Xc6cpU36X
Pr+kcsWYsztRiRFLdyOTyVwC+GRQR83oIMhaf+Y4wnI</cipherValue>
          </CipherData>
```

Figure C.8. (cont.) SOAP Message Example For Service Request From Client to Service Web Service

(Cont. on next page)

```
            <encryptionMethod>RSA-AES</encryptionMethod>
          </KeyInfo>
          <authenticationEndTime>2008-05-
25T21:41:54.3125+03:00
     </authenticationEndTime>
<authenticationMethod>UsernamePassword
     </authenticationMethod>
          <authenticationBeginTime>2008-05-
25T21:41:53.640625+03:00</authenticationBeginTime>
       </AuthenticationStatement>
       <Signature>
         <SignedInfo>
           <signatureAlgorithm>SHA1</signatureAlgorithm>
         </SignedInfo>

<signatureValue>GD6xPMaN2W9+jM3wTvCbMjP3dqU+gh5ldUxOS6vdERt
X6VKuCE4szcf3Qom5Mv9JL5GPm6+0+badyrKb2iDB4AqQMj8lVyOB9buPcf
0SoeQM4mhbINIIGiTNmvdRfghpqrKhDAUVZHZ7z/xPvXMvSzUtYDs86+xwJ
iEudI9zDp0=</signatureValue>
       </Signature>
       <issueInstant>2008-05-
25T21:41:54.46875+03:00</issueInstant>
       <issuer>Client</issuer>
       <KeyExchangeToken>
<encryptedSessionKey>rWTnYAhfewJP8vOiMDPLn9MbdIO1AosQTOp759
5hWy6JhfUkQ2MfYgWjPsLEj8thLDPRAd2WmaJl5QkhqkFFKoO9WN2anuMgx
wTDNKRu6Djk28n2zfNfIiwKk2PT5DsDZYGiLyAfDQDSEMHtBU1DAeNZsIzb
hKjicm9kbwlOH/I=</encryptedSessionKey>
<symmetricAlgorithmType>AES</symmetricAlgorithmType>
```

Figure C.8. (cont.) SOAP Message Example For Service Request From Client to Service Web Service

(Cont. on next page)

```
<encryptedSessionKeyIV>gy9txox+UQRQMPTQWM4SgyjQOtX1TPD3KH9w
tnkRxXl5sBKQKTXTo/aWqc3o5D5x+PpfT9rCWEHjk+OJuf7olEjSnlOKJCE
QKnfQHRKAWU8bhQ++mrIxyMtn1Ax/19/Hxh3a4x/jFzGA19mEU7E6IScPCk
8lTjDwZn3DLeCRlyY=</encryptedSessionKeyIV>
         <symmetricKeySize>64</symmetricKeySize>
         <asymmetricKeySize>1024</asymmetricKeySize>
         <KeyInfo>
           <encryptionMethod>RSA-AES</encryptionMethod>
         </KeyInfo>
       </KeyExchangeToken>
       <CreationTime>
         <TokenCreationBeginTime>2008-05-
25T21:41:54.3125+03:00</TokenCreationBeginTime>
         <TokenCreationEndTime>2008-05-
     25T21:41:54.46875+03:00</TokenCreationEndTime>
       </CreationTime>
     </CipherToken>
   </serializedCipherToken_IDToken>
      <serializedCipherToken_CreditStatusToken>
      <CipherToken
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Conditions>
    <notBefore>2008-05-25T21:41:56.28125+03:00</notBefore>
    <notOnOrAfter>2008-05-
25T21:56:56.28125+03:00</notOnOrAfter>
  </Conditions>
  <AuthenticationStatement>
    <KeyInfo>
```

Figure C.8. (cont.) SOAP Message Example For Service Request From Client to Service Web Service

(Cont. on next page)

```
 <CipherData>
<cipherValue>y2Y67tnzZzTXNGvZQsU2EI4sKbc32/cJrZkf/p2/U0AMpr
2OeH50HcgQzZielTODCjGL8M+/y2iaZXFnxNSR7bLiD3BteytXP0LujYBZk
Pkdfr9JqtbSRjB1vs83uGF2kAVZ6A82ByEQTD4hr7te0l/QGJHP0CkqBId1
aLWmb5Knm5N5EWc2aWziWiEV9M7jIikZh9SUsGJH6qvjgmXOAxUYLH5qlfY
qJReFioKVi6K0cv9o6q0o6I9yqWbPo+7YWxQyXahNEO4KcMFH5H41UuJhc1
42rJG1PiDTFvfB+lzvy0jIGmUQEc06ld6f4DmKuTAHKKEthCbMCwUiWH3MF
KRBR5sNd7ymhR0qJGZAahO/bfWtCNyrO5hPTJOWlnULBpJPVIDkifJOhLVu
1agyGKrs/z3HGfmG17rTEleU6vfjaQ+y7EOcSGmTsHvhJ0C+</cipherVal
ue>      </CipherData>
      <encryptionMethod>RSA-AES</encryptionMethod>
    </KeyInfo>
    <authenticationEndTime>2008-05-
25T21:41:55.984375+03:00</authenticationEndTime>
<authenticationMethod>SignatureAuthentication</authenticati
onMethod>
    <authenticationBeginTime>2008-05-
25T21:41:55.453125+03:00</authenticationBeginTime>
  </AuthenticationStatement>
  <Signature>
    <SignedInfo>
      <signatureAlgorithm>SHA1</signatureAlgorithm>
    </SignedInfo>
<signatureValue>qt/oNbCHxYTIrLl/F/vuaE4DuL6LgCsBxsOGXZj4QXH
R/a7Q2xESaFD2ihBAs6c5yeh3YrtNYjZXasSorbJem5sRMIlnHRuGt7RizE
5F63Wd+XOf88mElilPIooBl0of15T/eDtpFxK5TJ26r5v47nTNInokv0kwK
vDJwAF6FVE=</signatureValue>
  </Signature>
  <issueInstant>2008-05-
```

Figure C.8. (cont.) SOAP Message Example For Service Request From Client to Service Web Service

(Cont. on next page)

```
25T21:41:56.28125+03:00</issueInstant>
  <issuer>CreditStatusSTS</issuer>
  <CreationTime>
    <TokenCreationBeginTime>2008-05-
25T21:41:55.453125+03:00</TokenCreationBeginTime>
    <TokenCreationEndTime>2008-05-
25T21:41:56.28125+03:00</TokenCreationEndTime>
  </CreationTime>
</CipherToken>
     </serializedCipherToken_CreditStatusToken>
    </request>
  </soap:Body>
</soap:Envelope>
```

Figure C.8. (cont.) SOAP Message Example For Service Request From Client to Service Web Service