

WEB SERVICES CRYPTOGRAPHIC PATTERNS

By

Keiko Hashizume

A Thesis Submitted to the Faculty of
The College of Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Florida Atlantic University

Boca Raton, Florida

August 2009

WEB SERVICES CRYPTOGRAPHIC PATTERNS

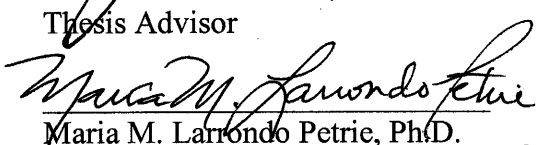
By

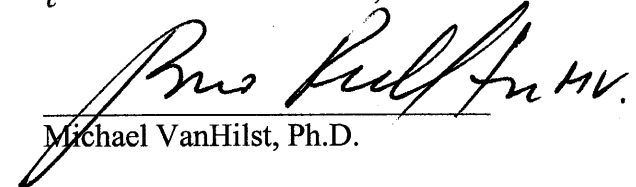
Keiko Hashizume

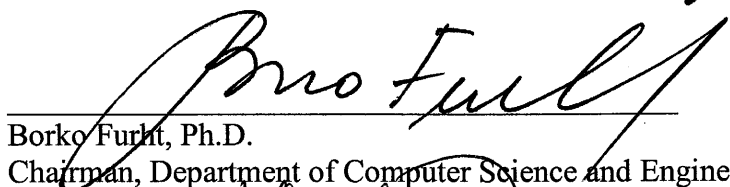
This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Eduardo B. Fernandez, Department of Computer Science and Engineering, and has been approved by the members of her supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

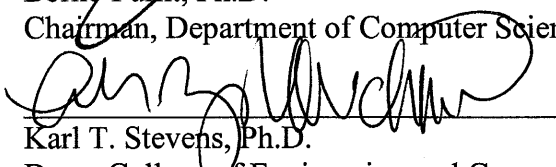
SUPERVISORY COMMITTEE:

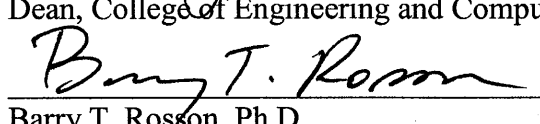

Eduardo B. Fernandez, Ph.D.
Thesis Advisor


Maria M. Larrondo Petrie, Ph.D.


Michael VanHilst, Ph.D.


Borko Furht, Ph.D.
Chairman, Department of Computer Science and Engineering


Karl T. Stevens, Ph.D.
Dean, College of Engineering and Computer Science


Barry T. Rosson, Ph.D.
Dean, Graduate College


Date

ACKNOWLEDGEMENTS

I would like to thank my advisor, Prof. Eduardo Fernandez, for his guidance during my research and study at Florida Atlantic University. I also want to express my gratitude to the members of the Secure Systems Research group for the constructive comments and suggestions they provided in developing the patterns. My studies were supported by the US Department of Defense (Secure Telecommunications Network) and by Pronto Progress, I thank both institutions.

ABSTRACT

Author: Keiko Hashizume
Title: Web Services Cryptographic Patterns
Institution: Florida Atlantic University
Thesis Advisor: Dr. Eduardo B. Fernandez
Degree: Master of Science
Year: 2009

Data security has been identified as one of the most important concerns where sensitive messages are exchanged over the network. In web service architecture multiple distributed applications communicate with each other over the network by sending XML messages. How can we protect these sensitive messages? Some web services standards have emerged to tackle this problem. The XML Encryption standard defines the process of encrypting and decrypting all of an XML message, part of an XML message, or even an external resource. Like XML Encryption, the XML Signature standard specifies how to digitally sign an entire XML message, part of an XML message, or an external object. WS-Security defines how to embed security tokens, XML encryption, and XML signature into XML documents. It does not define new security mechanisms but leverages existing security technologies such as encryption and digital signature.

WEB SERVICES CRYPTOGRAPHIC PATTERNS

| | |
|--|----|
| List of Table..... | ix |
| List of Figures..... | x |
| Introduction..... | 1 |
| Background..... | 4 |
| SOA..... | 4 |
| Web Services and Standards..... | 5 |
| The Current Status of Web Services Standards..... | 7 |
| Web Services Standards Classification..... | 7 |
| XML Specifications..... | 7 |
| Messaging Specifications..... | 9 |
| Description and Discovery Specifications..... | 11 |
| Security Specifications..... | 13 |
| Reliable Messaging Specifications..... | 17 |
| Business Process Specifications..... | 18 |
| Transaction Specifications..... | 18 |
| Management Specifications..... | 19 |
| Web Services Security Standards Interdependencies..... | 20 |
| Encryption Patterns..... | 23 |
| Symmetric Encryption..... | 24 |

| | |
|----------------------------|----|
| Intent..... | 24 |
| Example..... | 24 |
| Context..... | 25 |
| Problem..... | 25 |
| Solution..... | 25 |
| Implementation..... | 30 |
| Known Uses..... | 31 |
| Consequences..... | 32 |
| Example Resolved..... | 33 |
| Related Patterns..... | 33 |
| Asymmetric Encryption..... | 33 |
| Intent..... | 33 |
| Example..... | 34 |
| Context..... | 34 |
| Problem..... | 34 |
| Solution..... | 35 |
| Implementation..... | 40 |
| Known Uses..... | 41 |
| Consequences..... | 41 |
| Example Resolved..... | 43 |
| Related Patterns..... | 43 |
| XML Encryption..... | 44 |
| Intent..... | 44 |

| | |
|--------------------------------------|----|
| Example..... | 44 |
| Context..... | 44 |
| Problem..... | 44 |
| Solution..... | 45 |
| Implementation..... | 52 |
| Known Uses..... | 54 |
| Consequences..... | 54 |
| Related Patterns..... | 55 |
| Summary..... | 56 |
| Signature Patterns..... | 57 |
| Digital Signature with Hashing | 58 |
| Intent..... | 58 |
| Example..... | 59 |
| Context..... | 59 |
| Problem..... | 60 |
| Solution..... | 60 |
| Implementation..... | 67 |
| Known Uses..... | 69 |
| Consequences..... | 69 |
| Example Resolved..... | 71 |
| Related Patterns..... | 71 |
| XML Signature..... | 72 |
| Intent..... | 72 |

| | |
|---------------------------------|-----|
| Example..... | 72 |
| Context..... | 73 |
| Problem..... | 73 |
| Solution..... | 74 |
| Implementation..... | 83 |
| Known Uses..... | 84 |
| Consequences..... | 85 |
| Example Resolved..... | 87 |
| Related Patterns..... | 87 |
| Summary..... | 88 |
| WS-Security Pattern..... | 89 |
| Intent..... | 90 |
| Context..... | 90 |
| Problem..... | 90 |
| Solution..... | 91 |
| Implementation..... | 96 |
| Known Uses..... | 97 |
| Consequences..... | 97 |
| Related Patterns..... | 98 |
| Summary..... | 98 |
| Conclusion and Future Work..... | 99 |
| Reference..... | 100 |

TABLES

| | |
|---|----|
| Table 1. List of XML Specifications..... | 8 |
| Table 2.. List of Messaging Specifications..... | 9 |
| Table 3. .List of Description and Discovery Specifications..... | 11 |
| Table 4. .List of Security Specifications..... | 13 |
| Table 5. .List of Reliable Messaging Specifications..... | 17 |
| Table 6. .List of Business Process Specifications..... | 18 |
| Table 7. .List of Transaction Specifications..... | 18 |
| Table 8. .List of Management Specifications..... | 19 |

FIGURES

| | |
|---|----|
| Figure 1. Pattern Diagram for Web Services Security Standards..... | 22 |
| Figure 2. Class Diagram for Symmetric Encryption Pattern..... | 27 |
| Figure 3. Sequence Diagram for encrypting a message..... | 29 |
| Figure 4. Sequence Diagram for decrypting an encrypted message..... | 30 |
| Figure 5. Class Diagram for Symmetric Encryption Pattern..... | 37 |
| Figure 6. Sequence Diagram for encrypting a message..... | 39 |
| Figure 7. Sequence Diagram for decrypting an encrypted message..... | 40 |
| Figure 8. Class Diagram for XML Encryption Pattern..... | 49 |
| Figure 9. Sequence Diagram for encrypting XML elements..... | 50 |
| Figure 10. Sequence Diagram for decrypting XML elements..... | 51 |
| Figure 11. Class Diagram for Digital Signature Pattern..... | 64 |
| Figure 12. Sequence Diagram for signing a message..... | 66 |
| Figure 13. Sequence Diagram for verifying a signature..... | 67 |
| Figure 14. Class Diagram for XML Signature Pattern..... | 80 |
| Figure 15. Sequence Diagram for signing different XML elements..... | 81 |
| Figure 16. Sequence Diagram for verifying an XML Signature..... | 83 |
| Figure 17. Class diagram for WS-Security Pattern..... | 93 |
| Figure 18. Sequence Diagram for encrypting a message..... | 95 |
| Figure 19. Sequence Diagram for signing a XML element..... | 96 |

1. INTRODUCTION

Web services are components that are located in the Internet and can be incorporated into applications or as a standalone services. Web services are an alternative way for businesses to communicate with other businesses and also with clients. Web services communicate using XML messages that may contain sensitive data. How can we protect this data? Traditional protocols such as SSL and IPSec can be used to transport web services, but using these transport protocols lead to some limitations. SSL protects the data while they are in transit. After the data is delivered, the security is lost. Additionally, in secure transport layers, the entire message is protected. We cannot protect only the sensitive data; we cannot allow either different access to different parts of a document. In response of this deficiency, some standards have emerged to fill this gap.

XML Encryption and XML Signature are two of the basic standards in securing web services, and these standards are used by other emerging standards such as WS-Security. The XML Encryption standard defines the process of encrypting and decrypting all of an XML message, part of an XML message, or even an external resource. Encryption provides message confidentiality by protecting messages from being read by people other than the intended recipients. Like XML Encryption, the XML Signature standard specifies how to digitally sign an entire XML message, part of an XML

message, or an external object. This security mechanism, digital signature, provides message integrity (the message has not been changed since it was created) and message authentication (the message was originated from the sender). WS-Security defines how to embed XML encryption and XML signature into XML documents. It also defines how to embed security tokens such as Kerberos Tickets and X.509 which provide message authentication. WS-Security does not define new security mechanisms but leverages existing security technologies such as encryption and digital signature.

The problem with web services standards is that they can be lengthy documents that have too many details that makes difficult for vendors to develop products and for users to decide what product to use. Also, several organizations that have different goals have developed standards that may overlap and even conflict to each other. Thus, we develop patterns for these standards to have a better understanding of them. A pattern is an encapsulated solution to a recurrent problem. Patterns are described using a template. For this work we follow the POSA template [Bus96]. We develop some patterns that are used in SOA; however, we realized that these standards are quite complicated, so we also develop their abstract patterns that will describe how these mechanisms work in general.

Chapter 2 presents the reader background information that will be useful for the reader to understand better this work. In chapter 3, we present our classification of Web Services Standards, used as a reference to relate our patterns. Chapter 4 presents the XML Encryption Pattern and its abstract pattern Symmetric Encryption. Chapter 5 illustrates XML Signature Pattern and its abstract pattern Digital Signature with Hashing.

Chapter 6 presents the WS-Security Pattern. In chapter 7, we present some conclusions and possible future work.

2. BACKGROUND

This section provides basic concepts in order to have a better understanding of this work. We present a definition of SOA (Service Oriented Architecture) and web services and its standards.

2.1 Service Oriented Architecture (SOA)

Service Oriented Architecture (SOA) defines how entities communicate with each other, where one entity (service provider) performs some work on behalf of another entity (service user). A service represents a group of logical business operations. One important property of these services is that they are loosely coupled which minimizes the impact of change and allows convenient interoperability. SOA provides platform-independent enabling components to be implemented in different platforms, technologies, and languages. For example, a service can be implemented in .C#, and the application that consumes the services can be implemented on a different language.

In order to achieve interoperability, services register their descriptions such as interfaces and requirements that need to be met in order to communicate with them, using a specialized language, WSDL.

SOA can be also implemented using ad hoc architectures, web services, Jini, CORBA,

and others. However, the most common implementation of SOA is web services.

2.2 Web Services and Standards

Web service is defined by the W3C as “a software system designed to support interoperable machine-to-machine interaction over a network”. Web services define a set of operations available over the Internet. There are several organizations that are involved in the evolution of Web Services Standards, but there are three of them that are the key to the evolution of them: W3C (World Wide Web), OASIS (Organization for the Advancement of Structured Information Standards), and the WS-I Organization.

The primary goal of web services is to achieve universal interoperability between diverse systems by means of common standards. Four standards form the basis of web services: eXtensible Markup Language (XML), Web Services Description Language (WSDL), Universal Description, Discovery, and Integration (UDDI), and SOAP (Simple Object Access Protocol). XML is a W3C recommendation, and it is the foundation of all web services. It is a self-descriptive markup language that facilitates the exchange of structured information. WSDL is an XML-based standard that describes a set of operations that a web service provides, where the service is located, what services it can perform, and how to invoke it. UDDI is an XML-based language where businesses publish their web services so they can be discovered. SOAP is the communication protocol for exchanging XML messages.

There are a large number of web services standards. Security, reliability, and interoperability standards are some examples of web services standards that can be used in combination with the basic standards. Security standards such as WS-Security, XML Encryption, XML Signature, and other describe how to secure communication between applications through integrity, confidentiality, authentication, and authorization. WS-Reliability and WS-ReliableMessaging describes standards to guarantee the delivery of messages even in the presence of network failures.

3. THE CURRENT STATUS OF WEB SERVICES STANDARDS

We have classified these web services standards into eight groups: XML, Messaging, Description and Discovery, Security, Reliable Messaging, Business Process, Transaction, and Management Specifications. Each group identifies several standards that have similar objectives.

There are some standards that are composed by many parts such as XML Schema that has three parts: primer, structure, and data types. Usually the primer contains basic information to have a better understanding of the standard. The second part may be the framework or core that includes the main structure of the standard, and the other parts may be extended features. There are other standards that depend on others such as WS-Security that uses XML Encryption and XML Digital Signature. Even some other standards may overlap or conflict with each other such as ebXML and UDDI standards that define similar functionalities.

3.1. Web services Standards Classification

The following tables summarize the current web services standards.

3.1.1 XML Specifications

XML Specifications provides information about XML such as structure, schema, and namespaces. XML has also extended specifications that complement XML functionalities.

Table 1
List of XML Specifications

| Standard | Date | Publisher | Status | Description | Source |
|--|--------------|-----------|----------------|---|--------|
| XML 1.1 (eXtensible Markup Language) | Set 2006 | W3C | Recommendation | It is derived from SGML. It allows its users to create their own tags, enabling the definition, transmission, validation and interpretation of data between applications and between organizations [Xml06]. | W3C |
| XML Namespaces | Aug 2006 | W3C | Recommendation | They provide a simple method for qualifying element and attribute names used in XML documents by associating them with namespaces identified by URI references [Nam06]. | W3C |
| XML Schema Part 0: Primer | Oct 2004 | W3C | Recommendation | It is a non-normative document intended to provide an easily readable description of the XML Schema facilities [Sch04a]. | W3C |
| XML Schema Part 1: Structures | Oct 2004 | W3C | Recommendation | It offers facilities for describing the structure and constraining the contents of XML documents [Sch04b]. | W3C |
| XML Schema Part 2: Datatypes | Oct 2004 | W3C | Recommendation | It defines facilities for defining datatypes to be used in XML Schemas as well as other XML specifications [Sch04c]. | W3C |
| XPath 2.0 | Jan 2007 | W3C | Recommendation | It is a language for addressing parts of an XML document [Pat07]. | W3C |
| XQuery | Jan 2007 | W3C | Recommendation | It is a query language that is designed to query collections of XML data [Que07]. | W3C |
| XML Information Set | Feb 2004 | W3C | Recommendation | It provides a set of definitions for use in other specifications that need to refer to the information in an XML document [Inf04]. | W3C |
| XInclude | Nov 2006 | W3C | Recommendation | This specification introduces a generic mechanism for merging XML documents (as represented by their information sets) for use by applications that need such a facility [Inc06]. | W3C |
| XLink | June 2001 | W3C | Recommendation | It allows elements to be inserted into XML documents | W3C |

| | | | | | |
|----------------------------|------------|-----|----------------|---|-----|
| | | | | in order to create and describe links between resources [Lin01]. | |
| XPointer Framework | March 2003 | W3C | Recommendation | The framework is intended to be used as a basis for fragment identifiers for any resource [Poi03a]. | W3C |
| XPointer xmlns() Scheme | March 2003 | W3C | Recommendation | It is intended to be used with XPointer Framework to allow correct interpretation of namespace prefixes in pointers [Poi03b]. | W3C |
| XPointer xpointer() Scheme | Dec 2002 | W3C | Working Draft | It is intended to be used with the XPointer Framework to provide a high level of functionality for addressing portions of XML documents [Poi03c]. | W3C |

3.1.2 Messaging Specifications

This group includes specifications that enable entities to exchange XML messages in a distributed environment.

Table 2

List of Messaging Specifications

| Standard | Date | Publisher | Status | Description | Source |
|--------------------------------------|------------|-----------|----------------|---|------------|
| SOAP 1.2 Part 0: Primer | April 2007 | W3C | Recommendation | It is a non-normative document intended to provide an easily understandable tutorial on the features of SOAP Version 1.2 [Soap07a]. | W3C |
| SOAP 1.2 Part 1: Messaging Framework | April 2007 | W3C | Recommendation | It is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment [Soap07b]. | W3C |
| SOAP 1.2 Part 2: Adjuncts | April 2007 | W3C | Recommendation | It defines a set of adjuncts that MAY be used with the SOAP messaging framework such as SOAP encoding, SOAP RPC representation and so on [Soap07c]. | W3C |
| WS-Notification | Oct 2006 | OASIS | Standard | It is the base specification on which all the other specifications in the family depend. It consists of three specifications: WS- | IBM, OASIS |

| | | | | | |
|----------------------------------|-----------|-------|--------------------------|---|------------|
| | | | | BaseNotification, WS-BrokeredNotification and Ws-Topics. | |
| WS-BaseNotification | Oct 2006 | OASIS | Standard | It defines the normative Web services interfaces for two of the important roles: NotificationProducer and NotificationConsumer roles. It includes standard message exchanges to be implemented by service providers that wish to act in these roles, along with operational requirements expected of them [Not06a]. | IBM, OASIS |
| WS-BrokeredNotification | Oct 2006 | OASIS | Standard | It defines the Web services interface for the NotificationBroker. A NotificationBroker is an intermediary which, among other things, allows publication of messages from entities that are not themselves service providers [Not06b]. | IBM, OASIS |
| WS-Topics | Oct 2006 | OASIS | Standard | It defines a mechanism to organize and categorize items of interest for subscription known as “topics” [Top06]. | IBM, OASIS |
| WS-Addressing 1.0 - Core | May 2006 | W3C | Recommendation | It provides transport-neutral mechanisms to address Web services and messages. Specifically, this specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages [Add06a]. | W3C, IBM |
| WS-Addressing 1.0 – SOAP Binding | May 2006 | W3C | Recommendation | It defines the binding of the abstract properties defined in WS-Addressing 1.0 - Core to SOAP Messages [Add06b]. | W3C |
| WS-Addressing 1.0 - WSDL Binding | May 2006 | W3C | Candidate Recommendation | It defines how the abstract properties defined in WS-Addressing 1.0 - Core are described using WSDL [Add06c]. | W3C |
| WS-Addressing 1.0 - Metadata | Sept 2007 | W3C | Recommendation | It defines how the abstract properties defined in Web Services Addressing 1.0 - Core are described using WSDL, how to include WSDL metadata in endpoint references, and how WS-Policy can be used to indicate the support of WS-Addressing | W3C |

| | | | | | |
|--|------------|--------------------|-------------------|---|-----|
| | | | | by a Web service [Add06d]. | |
| WS-Transfer | Set 2006 | W3C | Member Submission | It describes a general SOAP-based protocol for accessing XML representations of Web service-based resources [Tra06]. | W3C |
| WS-Eventing | March 2006 | W3C | Member Submission | It describes a protocol that allows Web services to subscribe to or accept subscriptions for event notification messages [Eve06]. | IBM |
| WS-Enumeration | March 2006 | Microsoft, BEA, CA | Member Submission | It describes a general SOAP-based protocol for enumerating a sequence of XML elements that is suitable for traversing logs, message queues, or other linear information models [Enu06]. | W3C |
| SOAP Message Transmission Optimization Mechanism | Jan 2005 | W3C | Recommendation | It describes an abstract feature and a concrete implementation of it for optimizing the transmission and/or wire format of SOAP messages [Mtom05]. | W3C |

3.1.3 Description and Discovery Specifications

These specifications aim to describe Web Services in terms of location, operation, interfaces, and policies, and publish this information in order to be publicly accessed.

Table 3

List of Description and Discovery Specifications

| Standard | Date | Publisher | Status | Description | Source |
|---------------------------|----------|-----------|----------------|---|--------|
| WS-Policy 1.5 - Framework | Set 2007 | W3C | Recommendation | It provides a general purpose model and corresponding syntax to describe the policies of a Web Service [Pol07a]. | W3C |
| WS-PolicyAttachment 1.5 | Set 2007 | W3C | Recommendation | It defines two general-purpose mechanisms for associating policies, as defined in Web Services Policy 1.5 - Framework, with the subjects to which they apply. It also defines how these general-purpose mechanisms may be | W3C |

| | | | | | |
|--|------------|--------------------------------------|----------------|---|-----------|
| | | | | used to associate policies with WSDL and UDDI descriptions [Pol07b]. | |
| WS-Discovery | April 2005 | Microsoft, BEA, Intel | Draft | This specification defines a multicast discovery protocol to locate services [Dis05]. | Microsoft |
| WS-MetadataExchange 1.1 | Aug 2006 | BEA Systems, IBM, Microsoft, and SAP | Public Draft | Web services use metadata to describe what other endpoints need to know to interact with them [Met06]. | IBM |
| UDDI 3.0.2 (Universal Description, Discovery, and Integration) | Feb 2005 | OASIS | Standard | It defines a set of services supporting the description and discovery of (1) businesses, organizations, and other Web services providers, (2) the Web services they make available, and (3) the technical interfaces which may be used to access those services [Uddi05]. | OASIS |
| ebXML Registry Services and Protocols 3.0 | May 2005 | OASIS | Standard | It provides a set of services that enable sharing of content and metadata between organizational entities in a federated environment [Eb05a]. | OASIS |
| ebXML Registry: Information Model 3.0 | May 2005 | OASIS | Standard | It defines the types of metadata and content that can be stored in an ebXML Registry [Eb05b]. | OASIS |
| WSDL (Web Service Description Language) 1.1 | March 2001 | W3C | Note | It is an XML-based language for describing Web services and how to access them. It specifies the location of the service and the operations (or methods) the service exposes [Wsd01a]. | W3C |
| WSDL 2.0 Part 0: Primer | June 2007 | W3C | Recommendation | This primer is only intended to be a starting point toward use of WSDL 2.0, and hence does not describe every feature of the language [Wsd01b]. | W3C |
| WSDL 2.0 Part 1: Core | June 2007 | W3C | Recommendation | It defines the core language which can be used to describe Web services based on an abstract model of what the service offers. It also defines the conformance criteria for documents in this language [Wsd01c]. | W3C |
| WSDL 2.0 Part 2: Adjuncts | June 2007 | W3C | Recommendation | It specifies predefined extensions for use in WSDL 2.0: message exchange | W3C |

| | | | | | |
|---|------------|-------|--------------------|---|-------|
| | | | | patterns, operation safety, operation styles, and binding extensions for SOAP and HTTP [Wsd101d]. | |
| WSDL 2.0 SOAP 1.1 Binding | June 2007 | W3C | Working Group Note | It describes the concrete details for using WSDL 2.0 in conjunction with SOAP 1.1 protocol [Wsd101e]. | W3C |
| WSRF 1.2 Primer (WS-Resource Framework) | May 2006 | OASIS | Committee Draft | It defines a generic framework for modeling and accessing persistent resources using Web services [Res06a]. | OASIS |
| WS-Resource 1.2 | April 2006 | OASIS | Standard | It describes the relationship between a Web service and a resource in the WS-Resource Framework [Res06b]. | OASIS |
| WS-ResourceProperties 1.2 | April 2006 | OASIS | Standard | It standardizes the means by which the definition of the properties of a WS-Resource may be declared as part of a Web service interface [Res06c]. | OASIS |
| WS-ResourceLifetime 1.2 | April 2006 | OASIS | Standard | It defines two means of destroying a WS-Resource: immediate destruction and time-based, scheduled destruction [Res06d]. | OASIS |

3.1.4 Security Specifications

These security specifications describe how to secure communication between applications through integrity, confidentiality, authentication, and authorization.

Table 4

List of Security Specifications

| Standard | Date | Publisher | Status | Description | Source |
|--|------------|-----------|---------------|---|--------|
| AVDL 1.0 (Application Vulnerability Description Language) | May 2004 | OASIS | Specification | It describes a standard XML format that allows entities (such as applications, organizations, or institutes) to communicate information regarding web application vulnerabilities [Avdl04]. | OASIS |
| DSS 1.0 (Digital Signature Services) | April 2007 | OASIS | Standard | It defines the XML syntax and semantics for the Digital Signature Service core | OASIS |

| | | | | | |
|--|------------|-------|----------|--|-------|
| | | | | protocols, and for some associated core elements [Dsi07]. | |
| SAML 2.0 (Security Assertion Markup Language) Core | March 2005 | OASIS | Standard | It defines the syntax and semantics for XML-encoded assertions about authentication, attributes, and authorization, and for the protocols that convey this information [Sam105a]. | OASIS |
| SAML 2.0 (Security Assertion Markup Language) 2.0 Binding | March 2005 | OASIS | Standard | It defines protocol bindings for the use of SAML assertions and request-response messages in communications protocols and frameworks [Sam105b]. | OASIS |
| SAML 2.0 (Security Assertion Markup Language) 2.0 Profiles | March 2005 | OASIS | Standard | It defines profiles for the use of SAML assertions and request-response messages in communications protocols and frameworks, as well as profiles for SAML attribute value syntax and naming conventions [Sam105c]. | OASIS |
| SAML 2.0 (Security Assertion Markup Language) 2.0 Metadata | March 2005 | OASIS | Standard | It defines an extensible metadata format for SAML system entities, organized by roles that reflect SAML profiles [Sam105d]. | OASIS |
| SAML 2.0 (Security Assertion Markup Language) 2.0 Authentication Context | March 2005 | OASIS | Standard | It defines syntax for the definition of authentication context declarations and an initial list of authentication context classes for use with SAML [Sam105e]. | OASIS |
| SAML 2.0 (Security Assertion Markup Language) 2.0 Security and Privacy | March 2005 | OASIS | Standard | This non-normative specification describes and analyzes the security and privacy properties of SAML [Sam105f]. | OASIS |
| SPML 2.0 (Service Provisioning Markup Language) | April 2006 | OASIS | Standard | This specification defines the concepts and operations of an XML-based provisioning request-and-response protocol [Spml06]. | OASIS |
| WS-Security 1.1 Core | Feb 2006 | OASIS | Standard | It enhances SOAP messages in order to provide integrity and confidentiality. It also provides a general-purpose mechanism for associating security tokens with message | OASIS |

| | | | | | |
|--|-------------|-------|----------|--|-------|
| | | | | content [Sec04]. | |
| WS-Security: X.509 Certificate Token Profile 1.1 | Feb 2006 | OASIS | Standard | It describes how to use X.509 Certificates with the WS- Security [Cer06]. | OASIS |
| WS-Security: Username Token Profile 1.1 | Feb 2006 | OASIS | Standard | It describes how to use the Username Token with the WS-Security [Use06]. | OASIS |
| WS-Security: SAML Token Profile 1.1 | Feb 2006 | OASIS | Standard | It describes how to use Security Assertion Markup Language (SAML) V1.1 and V2.0 assertions with WS- Security [Saml06]. | OASIS |
| WS-Security: Kerberos Token Profile 1.1 | Feb 2006 | OASIS | Standard | It describes how to use Kerberos tickets (specifically the AP-REQ packet) with WS-Security [Ker06]. | OASIS |
| XACML 2.0 (Extensible Access Control Markup Language) Core | Feb 2005 | OASIS | Standard | It expresses policies for information access [Xacm05a]. | OASIS |
| XACML 2.0: Core and Hierarchical role based access control (RBAC) profile | Feb 2005 | OASIS | Standard | It defines a profile for the use of XACML in expressing policies that use role based access control (RBAC) [Xacm05b]. | OASIS |
| XACML 2.0: Hierarchical resource profile | Feb 2005 | OASIS | Standard | It provides a profile for the use XACML with resources that are structured as hierarchies [Xacm05c]. | OASIS |
| XACML 2.0: Multiple resource profile | Feb 2005 | OASIS | Standard | It provides a profile for requesting access to more than one resource in a single XACML Request Context, or for requesting a single response to a request for an entire hierarchy [Xacm05d]. | OASIS |
| XACML 2.0: Privacy policy profile | Feb 2005 | OASIS | Standard | It describes a profile of XACML for expressing privacy policies [Xacm05e]. | OASIS |
| XACML 2.0: SAML 2.0 profile | Feb 2005 | OASIS | Standard | It defines a profile for the use of SAML 2.0 to carry XACML 2.0 policies, policy queries and responses, authorization decisions, and authorization decision queries and responses. It also describes the use of SAML 2.0 Attribute Assertions with XACML [Xacm05f]. | OASIS |

| | | | | | |
|---|----------------|---|----------------|---|-------------|
| XACML 2.0: XML Digital Signature profile | Feb 2005 | OASIS | Standard | It uses XML-Signature Standard in order to provide authentication and integrity protection for XACML schema instances [Xacm05g]. | OASIS |
| XML Digital Signature | June 2008 | W3C | Recommendation | It specifies XML syntax and processing rules for creating and representing digital signatures [Sig08]. | W3C |
| XML encryption | Dec 2002 | W3C | Recommendation | It specifies a process for encrypting data and representing the result in XML [Enc02]. | W3C |
| XKMS 2.0 (XML Key Management Specification) | June 2005 | W3C | Recommendation | It specifies protocols for distributing and registering public keys, use in conjunction with the XML Signature [Xkms05a]. | W3C |
| XKMS 2.0 (XML Key Management Specification) Bindings | June 2005 | W3C | Recommendation | It specifies protocol bindings with security characteristics for the XKMS [Xkms05b]. | W3C |
| XrML 2.0 (Extensible Rights Management Language) | March 2002 | Content Guard | | It is based on XML and describes rights, fees and conditions together with message integrity and entity authentication information [Xrml02]. | XrML.org |
| XCBF 1.1 (XML Common Biometric Format) | August 2003 | OASIS | Standard | It defines XML codings for Common Biometric Exchange File Format [Bio03]. | OASIS |
| WS-Federation Language 1.1 | Dec 2006 | IBM, BEA, Microsoft, RSA, VeriSign, etc | Public Draft | Mechanisms to allow different security realms to federate. Allows brokering trust of identities, attributes, authentication between participating Web services [Fed06]. | IBM, BEA |
| WS-Federation: Active Requestor Profile 1.1 | July 2003 | IBM, BEA, Microsoft, RSA, VeriSign | Public Draft | It defines how federation mechanisms defined in WS- Federation are used by active requestors such as SOAP- enabled applications [Fed03a]. | IBM, BEA |
| WS-Federation: Passive Requestor Profile | July 2003 | IBM, BEA, Microsoft, RSA, VeriSign | Public Draft | It describes how WS- Federation can be utilized used by passive requestors such as Web browsers to provide Identity Services. Limited to the HTTP protocol [Fed03b]. | IBM, BEA |

| | | | | | |
|---------------------------|------------|-------|----------|--|-------|
| WS-SecureConversation 1.3 | March 2007 | OASIS | Standard | This specification defines extensions that build on [WS-Security] to provide a framework for requesting and issuing security tokens, and to broker trust relationships [Con07a]. | OASIS |
| WS-SecurityPolicy 1.2 | Jul 2007 | OASIS | Standard | It indicates the policy assertions for use with WS-Policy which apply to WS-Security, WS-Trust and WS-SecureConversation [Secp07]. | OASIS |
| WS-Trust 1.3 | March 2007 | OASIS | Standard | It defines extensions that build on WS-Security to provide a framework for requesting and issuing security tokens, and to broker trust relationships [Trus07]. | OASIS |

3.1.5 Reliable Messaging Specifications

These specifications guarantees the delivery of messages even when the system or network fails.

Table 5

List of Reliable Messaging Specifications

| Standard | Date | Publisher | Status | Description | Source |
|----------------------------|-----------|-----------|----------|--|--------|
| WS-ReliableMessaging 1.1 | June 2007 | OASIS | Standard | It describes a protocol that allows messages to be transferred reliably between nodes implementing this protocol in the presence of software component, system, or network failures [Rel07]. | OASIS |
| WS-Reliability 1.1 | Nov 2004 | OASIS | Standard | It is a SOAP-based protocol for exchanging SOAP messages with guaranteed delivery, no duplicates, and guaranteed message ordering [Rel04]. | OASIS |
| WS-RM Policy Assertion 1.1 | June 2007 | OASIS | Standard | It describes a domain-specific policy assertion for WS-ReliableMessaging that that can be specified within a policy alternative as defined in WS-Policy Framework [Rmp07]. | OASIS |

3.1.6 Business Process Specifications

Business Process Specifications are the highest level specifications that specify business process and participants involved in a transaction.

Table 6

List of Business Process Specifications

| Standard | Date | Publisher | Status | Description | Source |
|---|------------|------------------------|--------------------------|---|--------|
| WS-BPEL 2.0 | April 2007 | OASIS | Standard | It is a language for specifying business process behavior based on Web Services [Bpel07a]. | OASIS |
| Web Services Choreography Interface 1.0 | Aug 2002 | W3C, Sun, Intalio, BEA | Note | It is an XML-based interface description language that describes the flow of messages exchanged by a Web Service participating in choreographed interactions with other services [Cor05]. | W3C |
| WS-Choreography 1.0 | Nov 2005 | W3C | Candidate Recommendation | It is an XML-based language that describes peer-to-peer collaborations of participants by defining, from a global viewpoint, their common and complementary observable behavior; where ordered message exchanges result in accomplishing a common business goal [Wsci02]. | W3C |

3.1.7 Transaction Specifications

Transaction specifications provide coordination mechanisms when interoperability is needed between different domains.

Table 7

List of Transaction Specifications

| Standard | Date | Publisher | Status | Description | Source |
|---------------------|-----------|-----------|----------|--|--------|
| WS-Coordination 1.1 | July 2007 | OASIS | Standard | It describes an extensible framework for providing protocols that coordinate the | OASIS |

| | | | | | |
|--------------------------|------------|-------|----------|---|-------|
| | | | | actions of distributed applications [Coo07]. | |
| WS-BusinessActivity 1.1 | July 2007 | OASIS | Standard | It provides the definition of two Business Activity coordination types: AtomicOutcome or MixedOutcome, that are to be used with the extensible coordination framework described in the WS-Coordination specification [Bus07]. | OASIS |
| WS-AtomicTransaction 1.1 | April 2007 | OASIS | Standard | It provides the definition of the Atomic Transaction coordination type that is to be used with the extensible coordination framework described in WS-Coordination [Ato07]. | OASIS |
| WS-Context 1.0 | April 2007 | OASIS | Standard | It provides a definition, a structuring mechanism, and service definitions for organizing and sharing context across multiple execution endpoints [Con07]. | OASIS |

3.1.8 Management Specifications

These specifications describe how to manage and access web services or other resources located remotely on their networks.

Table 8

List of Management Specifications

| Standard | Date | Publisher | Status | Description | Source |
|-------------------------|-----------|--|---------------|--|--------|
| WS-Management 1.0 | Feb 2008 | DMTF | Specification | It describes a general SOAP-based protocol for managing systems such as PCs, servers, devices, Web services and other applications, and other manageable entities [Man08]. | DMTF |
| WS-Management Catalog | June 2005 | Intel, Dell, Microsoft, Sun and others | Specification | It describes the default metadata formats used for the WS-Management Protocol [Aro05]. | OASIS |
| WS-ResourceTransfer 1.0 | Aug 2006 | IBM, HP, Microsoft | Draft | It is intended to form an essential core component of a unified resource access | IBM |

| | | | | | |
|--|----------|-------|----------|---|-------|
| | | | | protocol for the Web services space [Rei06]. | |
| Management Using Web Services (MUWS)1.1 Part 1 | Aug 2006 | OASIS | Standard | It provides the fundamental concepts for management using Web services [Muws06a]. | OASIS |
| Management Using Web Services (MUWS)1.1 Part 2 | Aug 2006 | OASIS | Standard | It provides specific messaging formats used to enable the interoperability of MUWS implementations [Muws06b]. | OASIS |
| Management of Web Services (MOWS)1.1 | Aug 2006 | OASIS | Standard | It addresses management of the Web services endpoints using Web services protocols [Mows06]. | OASIS |

3.2 Web Services Security Standards Interdependencies

These web services security standards provide a way to communicate policy information or describe security mechanism different domains such as authentication, authorization, confidentiality, and integrity.

The XML Encryption standard describes a process to apply encryption functions to data but keeping a correct XML syntax. Likewise, the XML Signature provides a means to identify the source of the message (message authentication), and it provides also message integrity.

WS-Security defines how to secure SOAP messages applying XML security technologies such as XML Encryption and XML Signature. It also defines how to embed different security tokens. Security tokens provides authentication by proving one's identity.

WS-Policy describes how to express requirements that are needed or supported by a web service. For instance, it can indicate that a specific signature algorithm must be used when adding a digital signature.

SAML [Del07] defines a standard protocol to exchange authentication and authorization assertions. It may use WS-Security standard to protect assertions while they are being transmitted.

WS-Trust provides a framework for requesting and issuing security tokens, and to broker trust relationships [OAS07]. It uses WS-Security to transfer the required security tokens, using XML Signature and Encryption to ensure confidentiality. This standard may use WS-Policy to specify which security tokens are required at the target.

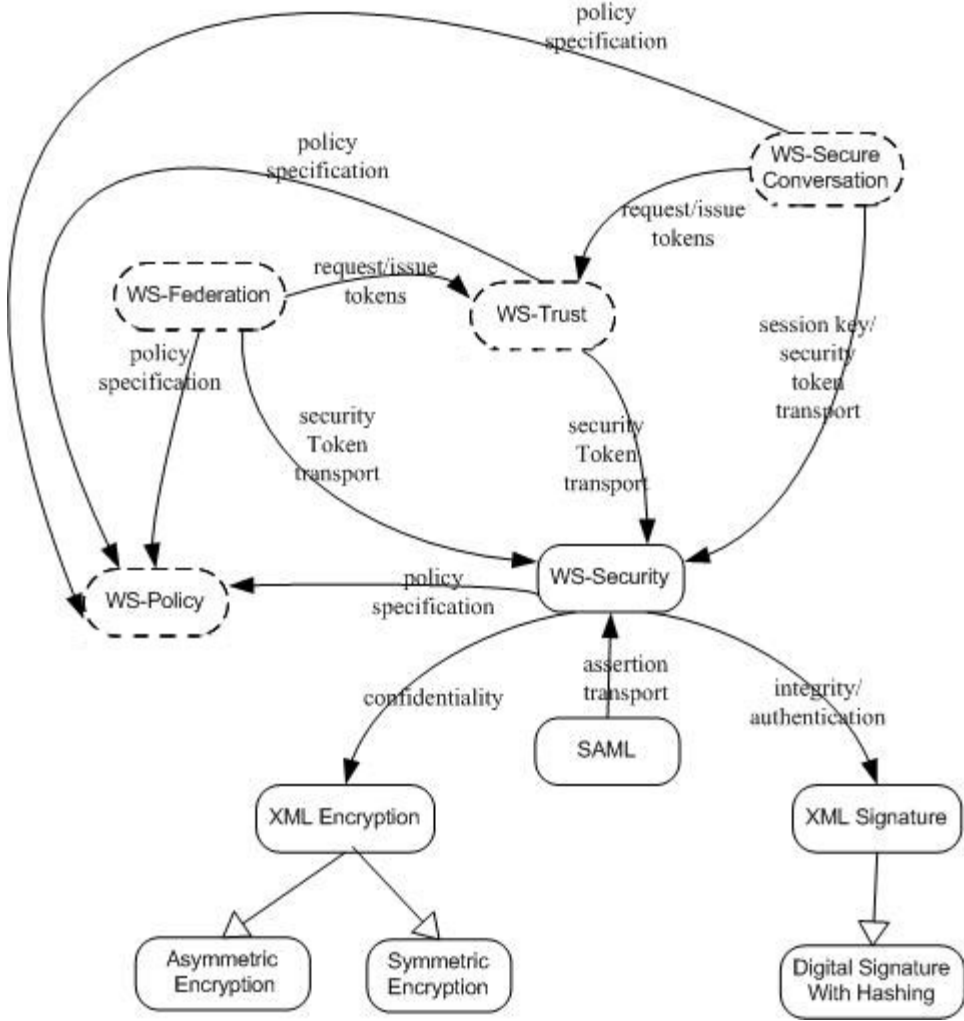
WS-SecureConversation defines mechanisms to allow security context establishment and sharing, and session key derivation [Con07a]. This specification uses WS-Security, WS-Trust and WS-Policy to negotiate and issue session keys.

WS-Federation defines mechanisms to allow different security domains to federate [Fed06]. It describes how federated trust scenarios can be constructed using WS-Security, WS-Policy, WS-Trust, and WS-SecureConversation.

Figure 1 illustrates the relationship between these web services security standards. The patterns that are represented in solid lines are described in this thesis or written earlier, and the ones in broken lines have not yet been written.

Figure 1

Patten Diagram for Web Services Security Standards



4. ENCRYPTION PATTERNS

An important security risk is that information can be captured and read during its transmission. How do we protect this information from being read by intruders? Encryption provides message confidentiality by transforming readable data (plain text) into an unreadable format (cipher text) that can be understood only by the intended receiver after a process called decryption, the inverse function that makes the encrypted information readable again. There are two types of encryption: symmetric and asymmetric encryption. In symmetric encryption a common key is used for both encryption and decryption. In asymmetric encryption a public/private key pair is used for encryption/decryption; the sender encrypts the information using the receiver's public key, while the receiver uses his private key to decrypt the ciphered text.

The encrypted messages may be intercepted and be the object of attacks, including illegal reading, modification, and replay. An emerging use of web services that exchanges XML messages also can be target of attacks. Some security standards have been developed to apply correctly encryption functions and thus reduce security risks. XML Encryption is one of the basic standards in securing web services. XML Encryption defines how to encrypt/decrypt an entire XML message, part of an XML message, or an external object linked to the message, and how to represent the encrypted content and information such as encryption algorithm and key in XML format. We present here

patterns for Symmetric Encryption and for XML Encryption. By presenting Symmetric Encryption first we make the second pattern easier to understand.

Section 2 presents the Symmetric Encryption Pattern, and Section 3 presents the XML Encryption pattern. We assume the reader is an application designer intending to use message secrecy in her design and has a basic knowledge of cryptography and UML. The XML pattern could also be of value to a designer of cryptographic products. While the XML pattern does not include all aspects of the standard it has sufficient detail so as it can be used as a guideline for design.

4.1 Symmetric Encryption

4.1.1 Intent

Encryption protects message confidentiality by making a message unreadable to those that do not have access to the key. Symmetric encryption uses the same key for encryption and decryption.

4.1.2 Example

Alice, in the Purchasing department regularly sends purchase orders to Bob in a distribution office. A purchase order contains sensitive data such as credit card numbers and other company information, so it is important to keep it secret. Eve can intercept her messages and may try to read them to get the confidential information.

4.1.3 Context

Applications that exchange sensitive information over insecure channels.

4.1.4 Problem

Applications that communicate with external applications interchange sensitive data that may be read by unauthorized users while they are in transit. How do we protect messages from being read by intruders?

The solution for this problem is affected by the following forces:

- *Confidentiality*--Messages may be captured while they are in transit, so we need to prevent unauthorized users from reading them by hiding the information of the message. Hiding the information also makes replaying of messages by an attacker harder to perform.
- *Reception*--The hidden information should be revealed conveniently to the receiver.
- *Protocol*--We need to apply the solution properly or it will not be able to stand attacks (there are several ways to attack a method to hide information).
- *Performance*--The time to hide and recover the message should be reasonable.

4.1.5 Solution

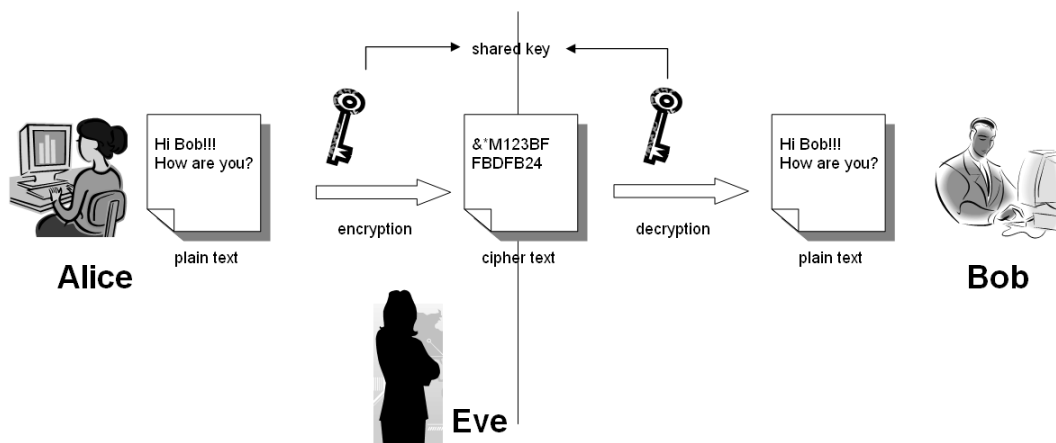
Transform a message in such a way that only can be understood by the intended receiver after applying the reverse transformation using a valid key. The transformation process at the sender's end is called Encryption, while the reverse transformation process at the receiver's end is called Decryption.

The sender applies an encryption function (E) to the message (M) using a key (k); the output is the cipher text (C).

$$C = E_k (M)$$

When the cipher text (C) is delivered, the receiver applies a decryption function (D) to the cipher text using the same key (k) and recovers the message, i.e.

$$M = D_k (C)$$

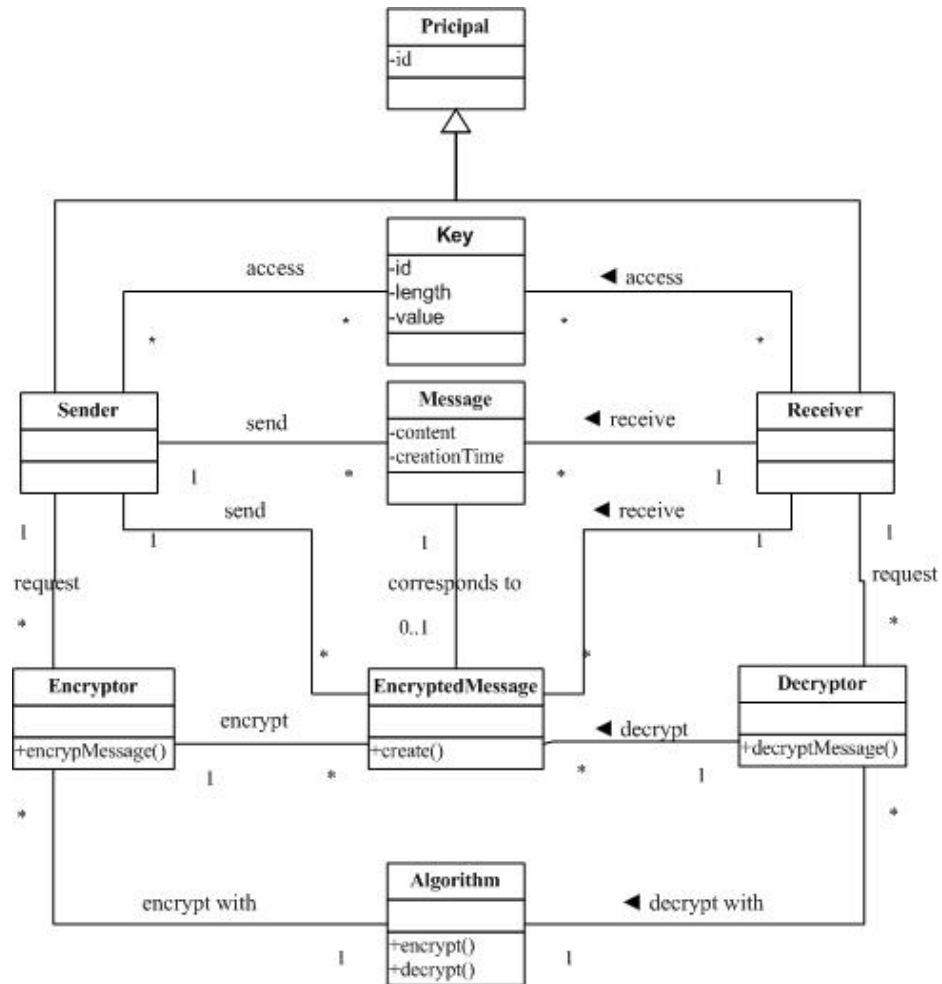


Structure

Figure 2 describes the class diagram for the Symmetric Encryption Pattern.

Figure 2

Class Diagram for Symmetric Encryption Pattern



A **Principal** may be a user or an organization that is responsible for sending or receiving messages. This **Principal** may have the roles of **Sender** or **Receiver**. A Sender may send a Message and/or an **EncryptedMessage** to a receiver with which it shares a secret **Key**. The **Encryptor** creates the EncryptedMessage that contain the cipher text using the shared key provided by the sender, while the **Decryptor** decipheres the encrypted data into

its original form using the same key. Both the Encryptor and Decryptor use the same **Algorithm** to encipher and decipher a message.

Dynamics

We describe the dynamic aspects of the Encryption Pattern using sequence diagrams for the following use cases: encrypt a message and decrypt a message.

Encrypt a message (Figure 3):

Summary: A Sender wants to encrypt a message

Actors: A Sender

Precondition: Both sender and receiver have a shared key and access to a repository of algorithms. The message has already been created by the sender.

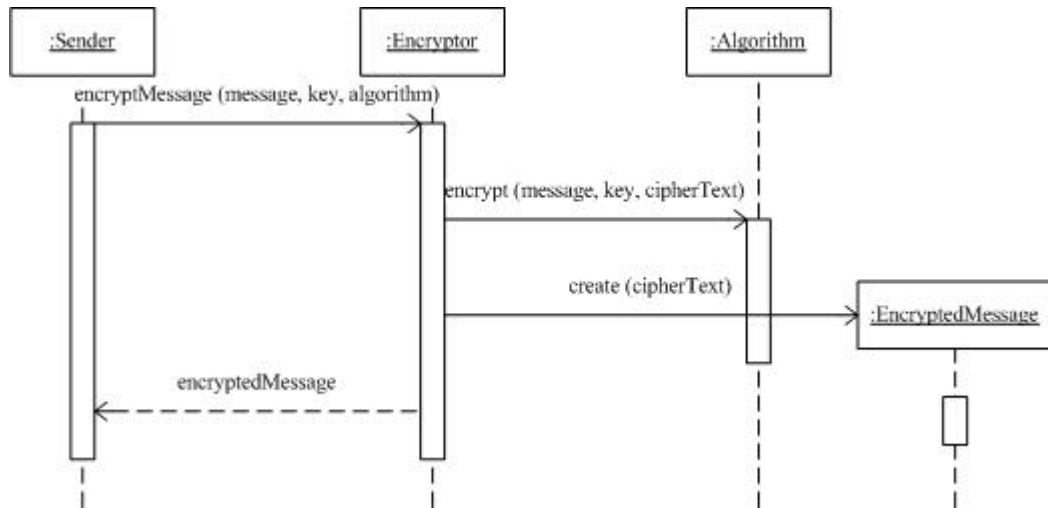
Description:

- a) A Sender sends the message, the shared key, and the algorithm identifier to the Encryptor.
- b) The Encryptor ciphers the message using the algorithm specified by the sender.
- c) The Encryptor creates the EncryptedMessage that includes the cipher text.

Postcondition: The message has been encrypted and sent to the sender.

Figure 3

Sequence Diagram for Encrypting a Message



Decrypt an Encrypted Message (Figure 4):

Summary: A receiver wants to decrypt an encrypted message from a sender.

Actors: A Receiver

Precondition: Both the sender and receiver have a shared key and access to a repository of algorithms.

Description:

- a) A Receiver sends the encrypted message and the shared key to the decryptor.
- b) The Decryptor decipheres the encrypted message using the shared key.
- c) The Decryptor creates the Message that contains the plain text obtained from the previous step.
- d) The Decryptor sends the plain Message to the receiver.

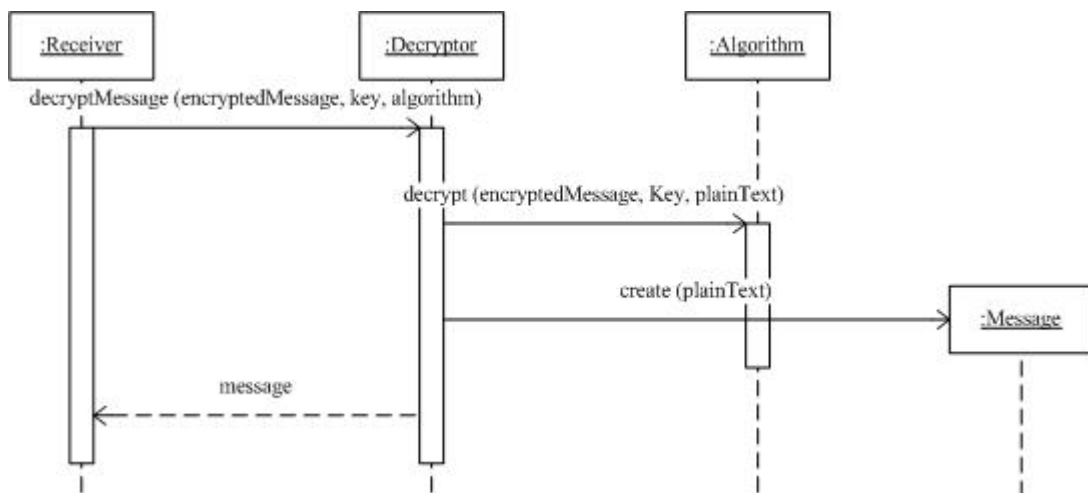
Alternate Flows:

- If the key used in step b) is not the same as the one used for encryption, the decryption process fails.

Postcondition: The encrypted message has been deciphered and delivered to the Receiver.

Figure 4

Sequence Diagram for Decrypting an Encrypted Message



4.1.6 Implementation

- Use the Strategy Pattern [Gam94] to select different encryption algorithms.
- The designer should choose well-known algorithms such as AES (Advanced Encryption Standard) [Fed01] and DES (Data Encryption Standard) [Fed99].
- Encryption can be implemented in different applications such as in email communication, distribution of documents over the Internet, or web services. In

these applications, we are able to encrypt the entire document. However, in web services we can encrypt parts of a message.

- Both the sender and the receiver have to previously agree what cryptographic algorithm they support.
- A good key generator is very important. It should generate keys that are as random as possible or an attacker who captures some messages could be able to deduce the key.
- A long encryption key should be used (at least 64 bits). Only brute force is known to work against the DES and AES for example; using a short key would let the attacker generate all possible keys.

4.1.7 Known Uses

Symmetric Encryption has been widely used in different products.

- GNUPG [Gnu] is free software that secures data from eavesdroppers.
- OpenSSL [Ope] is an open source toolkit that encrypts and decrypts files.
- Java Cryptographic Extension [Suna] provides a framework and implementations for encryption.
- The .NET framework [Mica] provides several classes to perform encryption and decryption using symmetric algorithms.
- XML Encryption [W3C02] is one of the foundation web services security standards that defines the structure and process of encryption for XML messages.
- Pretty Good Privacy (PGP), a set of programs used mostly for e-mail security, includes methods for symmetric encryption and decryption [PGP].

4.1.8 Consequences

This pattern presents the following advantages:

- Only receivers who possess the shared key can decrypt a message transforming it into a readable form. A captured message is unreadable to the attacker. This makes attacks based on replaying a message very hard.
- The strength of a cryptosystem is based on the secrecy of a long key [Sta06]. The cryptographic algorithms are known to the public, so the key should be kept protected from unauthorized users.
- It is possible to select from several encryption algorithms the one suitable for the application needs.
- There exist encryption algorithms that take a reasonable time to encrypt messages.

The pattern also has some (possible) liabilities:

- This pattern assumes that the shared key was distributed in a secure way. This may not be easy for large groups of nodes exchanging messages. Asymmetric cryptography can be used to solve this problem.
- Cryptography operations are computationally intensive and may affect the performance of the application. This is particularly important for mobile devices.
- Encryption does not provide data integrity. The encrypted data can be modified by an attacker, other means such as hashing, are needed to verify that the message was not changed.

- Encryption does not prevent a replay attack because an encrypted message can be captured and resent without being decrypted. It is recommended to use another security mechanism such as Timestamps or Nonce to prevent this attack.

4.1.9 Example resolved

Alice now encrypts the purchase orders she sends to Bob. The purchase's order sensitive data is now unreadable to Eve. Eve can try to apply to it all possible keys but if the algorithm has been well chosen and implemented, she cannot read the confidential information.

4.1.10 Related Patterns

- The Secure Channel Communication pattern [Bra98], supports the encryption/decryption of data. This pattern describes encryption in more general terms. It does not distinguish between asymmetric and symmetric encryption. Another version is given in [Sch06].
- Strategy Pattern [Gam94], defines how to separate the implementation of related algorithms from the selection of one of them. This pattern can be used to select an encryption algorithm dynamically.
- Asymmetric Encryption is commonly used to distribute keys.

4.2 Asymmetric Encryption

4.2.1 Intent

Encryption provides message confidentiality by keeping information secret in such a way that it can only be understood by intended recipients who have the access to the valid key. In asymmetric encryption, a public/private key pair is used for encryption and decryption respectively.

4.2.2 Example

Alice wants to send a personal message to Bob. They have not met each other to agree upon a shared key. She wants to keep the message secret since it contains personal information. Eve can intercept her messages and may try to obtain the confidential information.

4.2.3 Context

Applications that exchange sensitive information over insecure networks.

4.2.4 Problem

Applications that communicate with external applications interchange messages that may contain sensitive. These messages can be intercepted and read by impostors during transmission. How do can we send sensitive information securely over insecure channels?

The solution for this problem is affected by the following forces:

- *Confidentiality*--Messages may be captured while they are in transit, so we need to prevent unauthorized users from reading them by hiding the information of the

message. Hiding the information also makes replaying of messages by an attacker harder to perform.

- *Reception*--The hidden information should be revealed conveniently to the receiver.
- *Protocol*--We need to apply the solution properly or it will not be able to stand attacks (there are several ways to attack a method to hide information.
- *Performance*--The time to hide and recover the message should be reasonable.
- *Key distribution* -- Two parties may want to communicate to each other, but they have not agreed upon a shared key. Thus, we need a way to send messages without establishing a common key.

4.2.5 Solution

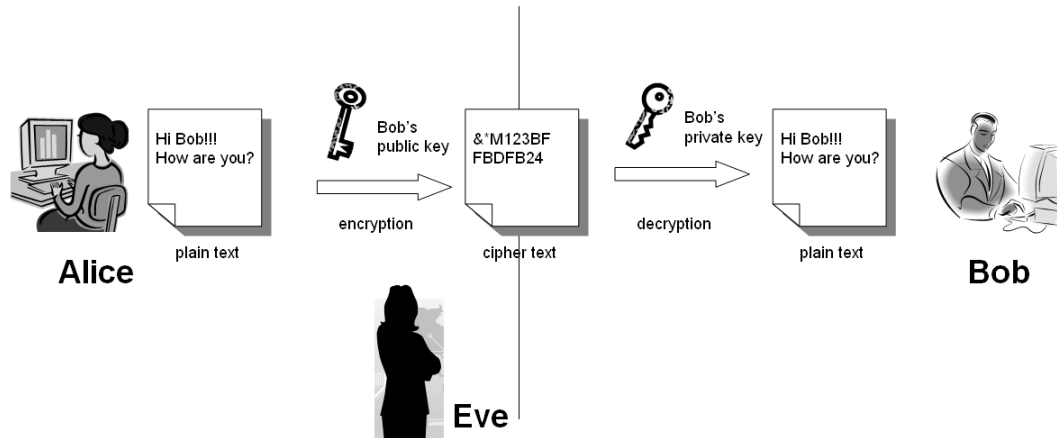
Apply mathematical functions to a message, so it can unreadable to those that do not have a valid key. This approach uses a key pair: private and public key.

The sender encrypts (E) the message (M) using the receiver's public key (PuK) that is accessible by anyone. The result of this process is cipher text (C)

$$C = E_{\text{PuK}}(M)$$

On the other side, the receiver decrypts (D) the cipher text (C) using his private key (PrK) to recover the plain message (M).

$$M = D_{PrK} (C)$$



Structure

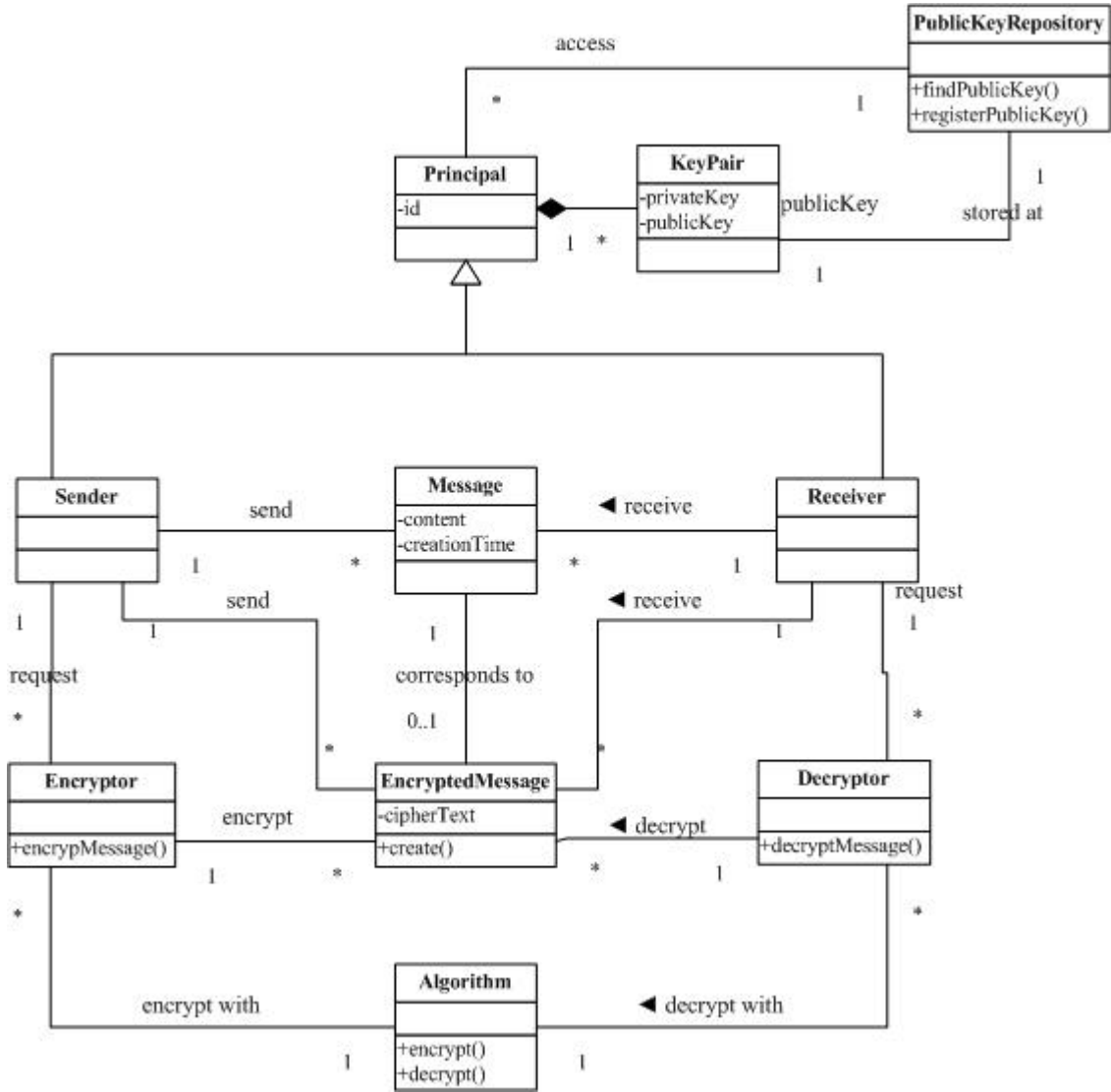
Figure 1 describes the class diagram for the Asymmetric Encryption Pattern.

A **Principal** may be a user or an organization that is responsible for sending or receiving messages. This **Principal** may have the roles of **Sender** or **Receiver**. A Sender may send a Message and/or a **EncryptedMessage** to a receiver with which it shares a secret **Key**.

A **Principal** has one or more **KeyPair** that is composed of a private key that is kept secret by its owner and a public key that is publicly published. **PublicKeyRepository** is a repository that contains a list of public keys where users can register and/or access public keys. These two keys are mathematically related, so while one encrypts, the other decrypts. However, it is not feasible to deduce one's private key from its corresponding public key.

Figure 5

Class Diagram for Asymmetric Encryption Pattern



The **Encryptor** creates the **EncryptedMessage** that contain the cipher text using the shared key provided by the sender, while the **Decryptor** decipheres the encrypted data into its original form using the same key. Both the **Encryptor** and **Decryptor** use the same **Algorithm** to encipher and decipher a message.

Dynamics

We describe the dynamic aspects of the Asymmetric Encryption Pattern using sequence diagrams for the following use cases: encrypt a message and decrypt a message.

Encrypt a message (Figure 2):

Summary: A Sender wants to encrypt a message.

Actors: A Sender

Precondition: The sender has access to the receiver's public key. Both sender and receiver have access to a repository of algorithms. The message has already been created by the sender.

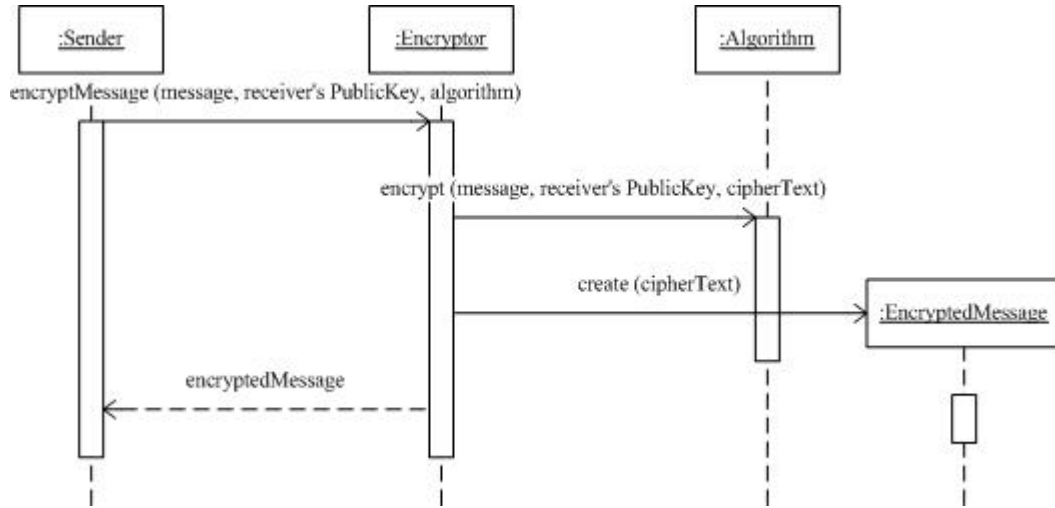
Description:

- d) A Sender sends the message, the receiver's public key, and the algorithm identifier to the Encryptor.
- e) The Encryptor ciphers the message using the algorithm specified by the sender.
- f) The Encryptor creates the EncryptedMessage that includes the cipher text.

Postcondition: The message has been encrypted and sent to the sender.

Figure 6

Sequence Diagram for Encrypting a Message



Decrypt an Encrypted Message (Figure 3):

Summary: A receiver wants to decrypt an encrypted message from a sender.

Actors: A Receiver

Precondition: Both the sender and receiver have access to a repository of algorithms.

Description:

- e) A Receiver sends the encrypted message and his private key to the decryptor.
- f) The Decryptor decipheres the encrypted message using the receiver's public key.
- g) The Decryptor creates the Message that contains the plain text obtained from the previous step.
- h) The Decryptor sends the plain Message to the receiver.

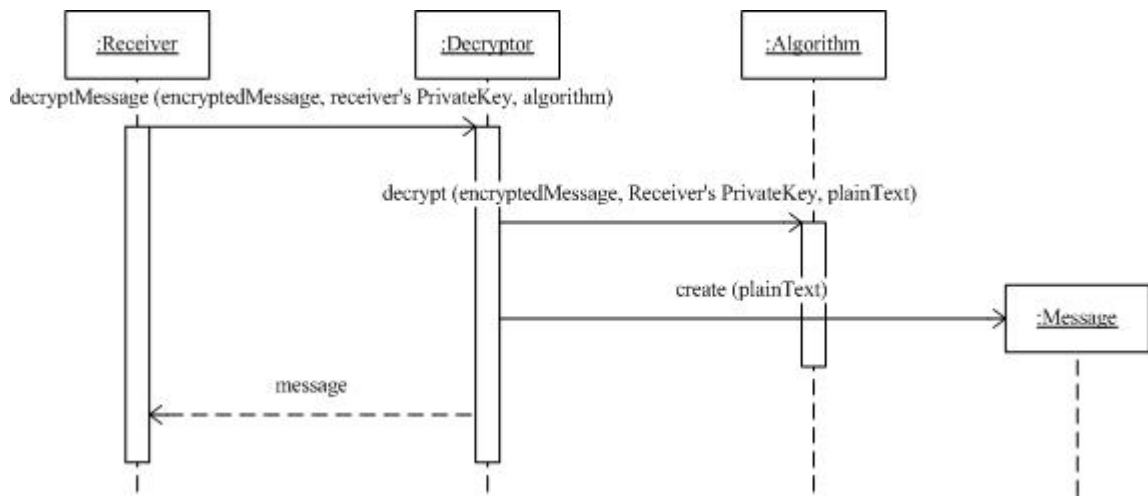
Alternate Flows:

- If the key used in step b) is not mathematically related to the key used for encryption, the decryption process fails.

Postcondition: The encrypted message has been deciphered and delivered to the Receiver.

Figure 7

Sequence Diagram for Decrypting an Encrypted Message



4.2.6 Implementation

- Use the Strategy Pattern [Gam94] to select different encryption algorithms.
- The designer should choose well-known algorithms such as RSA that was developed by Ronald Rivest, Adi Shami, and Len Adleman [Riv78].
- Encryption can be implemented in different applications such as in email communication, distribution of documents over the Internet, or web services. In

these applications, we are able to encrypt the entire document. However, in web services we can encrypt parts of a message.

- Both the sender and the receiver have to previously agree what cryptographic algorithm they support.
- A good key pair generator is very important. It should generate key pairs where the private key cannot be deduced from the public key.

4.2.7 Known Uses

Asymmetric Encryption has been widely used in different products.

- GNUPG [Gnu] is free software that secures data from eavesdroppers.
- Java Cryptographic Extension [Sun] supports a variety of algorithms including asymmetric encryption.
- The .NET framework [Mic] provides several classes to perform asymmetric encryption and decryption.
- XML Encryption [W3C02] is one of the foundation web services security standards that defines the structure and process of encryption for XML messages. This standard supports both types of encryption: symmetric and asymmetric encryption.
- Pretty Good Privacy (PGP) uses asymmetric encryption and decryption as one of its process to secure e-mail communication [PGP].

4.2.8 Consequences

This pattern presents the following advantages:

- Asymmetric encryption does not require a secret key to be shared among all the participants. Anyone can look up for the public key in the repository and send messages to the owner of the public key.
- Recipients that possess the corresponding private key can make the encrypted message readable again.
- The strength of a cryptosystem is based on the secrecy of a long key [Sta06]. The cryptographic algorithms are known to the public, so the key should be kept protected from unauthorized users.
- It is possible to select from several encryption algorithms the one suitable for the application needs.
- There exist encryption algorithms that take a reasonable time to encrypt messages.

The pattern also has some (possible) liabilities:

- Cryptography operations are computationally intensive and may affect the performance of the application. Asymmetric encryption is slower than symmetric encryption. Thus, it is recommended to use a combination of both algorithms: asymmetric encryption for key distribution and symmetric encryption for message exchanging.
- Encryption does not provide data integrity. The encrypted data can be modified by an attacker, other means such as hashing, are needed to verify that the message was not changed.

- Encryption does not prevent a replay attack because an encrypted message can be captured and resent without being decrypted. It is recommended to use another security mechanism such as Timestamps or Nonce to prevent this attack.
- This pattern assumes that a public key belongs to the person who he claims to be. How do we know that this person is not impersonating another one? To confirm that a person is who he says he is, we can use Certificates issued by some Certification Authority.

4.2.9 Example Resolved

Alice now can look up for Bob's public key and encrypts the message using this key. Since Bob keeps his private key secret, he is the only one who can decrypt the message. Eve cannot understand the encrypted data since she does not have access to Bob's private key.

4.2.10 Related Patterns

- The Secure Channel Communication pattern [Bra98], supports the encryption/decryption of data. This pattern describes encryption in more general terms. It does not distinguish between asymmetric and symmetric encryption. Another version is given in [Sch06].
- Strategy Pattern [Gam94], defines how to separate the implementation of related algorithms from the selection of one of them. This pattern can be used to select an encryption algorithm dynamically.

4.3 XML Encryption Pattern

4.3.1 Intent

The XML Encryption standard [W3C02] describes the syntax to represent XML encrypted data and the process of encryption and decryption. XML Encryption provides confidentiality by hiding selected sensitive information in a message using cryptography.

4.3.2 Example

Alice, in the Purchasing department regularly sends purchase orders in the form of XML documents to Bob, who works in a distribution office. The purchase order contains sensitive data such as credit card numbers and other company information, so it is important to keep it secret. In the receiving end, different people will handle different parts of the order. Eve can intercept these orders and may try to read them to get the confidential information.

4.3.3 Context

Users of web services send and receive SOAP messages through insecure networks such as the Internet.

4.3.4 Problem

In many applications that communicate with external applications the users interchange sensitive data. This data may be read by unauthorized people while the messages are in transit.

The solution for this problem is affected by the following forces:

- Messages may be captured while they are in transit, so we need to prevent unauthorized users from reading them by hiding the information of the message using encryption.
- We need to express encrypted elements in a standardized XML format to allow encrypted data to be nested within an XML message. Otherwise, different applications cannot interoperate.
- Different parts of a message may be intended for different recipients, and not all the information contained within a message should be available to all the recipients. Thus, recipients should be able to read only those parts of the message that are intended for them.
- For flexibility reasons, both symmetric and asymmetric encryption algorithms should be supported.
- If a secret key is embedded in the message, it should be protected. Otherwise, an attacker could read some messages.

4.3.5 Solution

Transform a message using some encryption algorithm so that it can only be understood by legitimate receivers that possess a valid key.

First, the data has to be serialized before encryption. The serialization process will convert the data into octets. Then, this serialized data is encrypted using the chosen algorithm and the encryption key. The cipher data and the information of the encryption (algorithm, key, and other properties) are represented in XML format.

XML Encryption supports both types of encryption: symmetric and asymmetric. The symmetric encryption algorithm uses a common key for both encryption and decryption. The asymmetric encryption algorithm uses a key pair (public key and private key). The sender encrypts a message using the receiver's public key, and the receiver uses its private key to decrypt the encrypted message. Thus, in both types of encryption, only recipients who possess the shared key or the private key that matches the public key used in the encryption process can read the encrypted message after decryption.

Structure

Figure 5 describes the structure of the XML Encryption Pattern. The yellow classes correspond to the classes of the Encryption pattern, the white classes describe the fact that encryption can now be applied to specific portions of the message.

A **Principal** may be a user or an organization that sends and receives **XMLMessages** and/or **EncryptedXMLMessages**. This principal may have the roles of **Sender** and **Receiver**.

Both an **XMLMessage** and a **EncryptedMLMessage** are composed of XML elements. Each XMLElement may have many children, and each child also can be composed by other XML elements, and so on. The **Encryptor** and the **Decryptor** encipher a message and decipher an encrypted message respectively.

The **EncryptedData** contains other subelements such as the encryption method, key information, cipher value, and encryption properties. The **EncryptionMethod** is an optional element that specifies the algorithm used to encrypt the data. If this element is not specified, the receiver must know the encryption algorithm. The **KeyInfo** (optional) contains the same key information as the one describes in the XML Signature standard [W3C08]. However, this standard defines two other subelements: **EncryptedKey** and **ReferenceList**. The EncryptedKey contains similar elements as the EncryptedData; however, they are not shown in the class diagram. The EncryptedKey includes an optional ReferenceList element that points to data or keys encrypted using this key. The **CipherData** is a mandatory element that stores either the cipher value or a pointer (cipher reference) where the encrypted data is located. The **EncryptionProperties** element holds information such as the time that the encryption was performed or the serial number of the hardware used for this process.

Dynamics:

We describe the dynamic aspects of the XML Encryption Pattern using sequence diagrams for the following use cases: “encrypt XML elements” and “decrypt an encrypted XML message”.

Encrypt XML elements (Figure 6):

Summary: A sender wants to encrypt different elements of an XML message using a shared key.

Actors: A sender

Precondition: Both sender and receiver have a shared key and a list of encryption algorithms.

Description:

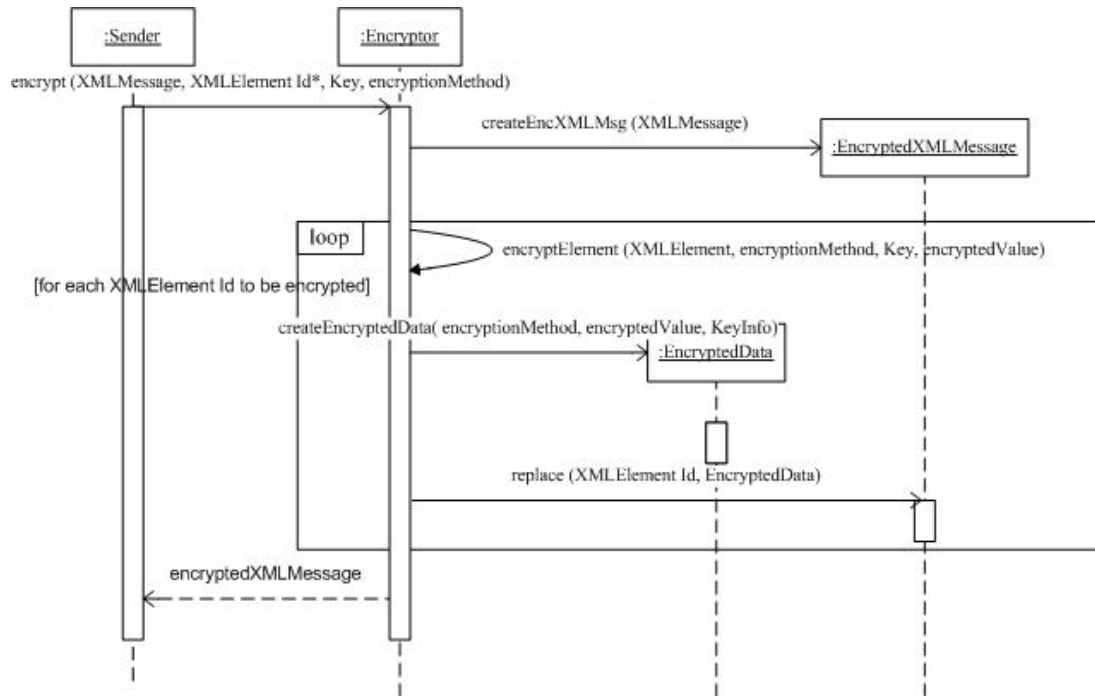
- a) A sender requests to the encryptor to encrypt a list of XML elements. This list is represented with an asterisk (*) in the sequence diagram.
- b) The encryptor creates the EncryptedXMLMessage.
- c) The encryptor encrypts the XML Element using the shared key and the encryption method provided by the sender and produces an encrypted value.
- d) The encryptor creates the EncryptionData element including the EncryptionMethod that holds the encryption algorithm used to encrypt the data, the KeyInfo that contains information about the key, and the CipherData obtained from step c)
- e) The encryptor replaces the XML element with the encrypted data.
- f) Repeat steps c) to e) for each XML element to encrypt.
- g) The encryptor sends the EncryptedXMLMessage to the sender.

Alternate Flows: none

Postcondition: The encrypted XML message has been created.

Figure 9

Sequence Diagram for encrypting XML Elements



Decrypt an Encrypted XML Message (Figure 7):

Summary: A receiver wants to decrypt an encrypted XML message.

Actors: A Receiver

Precondition: Both sender and receiver have a shared key and a list of encryption

algorithms Description:

- a) A receiver requests to the verifier to decrypt an encrypted XML message.
- b) The decryptor creates the XMLMessage that contains a copy of the EncryptedXMLMessage.
- c) The decryptor obtains the elements within the EncryptedData element such as the EncryptionMethod, KeyInfo, and the cipherValue.

- d) The decryptor decrypts the cipher value using the encryption method and the shared key.
- e) The decryptor replaces the encrypted data with the plain text obtained from the previous step.
- f) Repeat steps c) to e) for each XML element to decrypt.
- g) The decryptor sends the decrypted XMLMessage to the receiver.

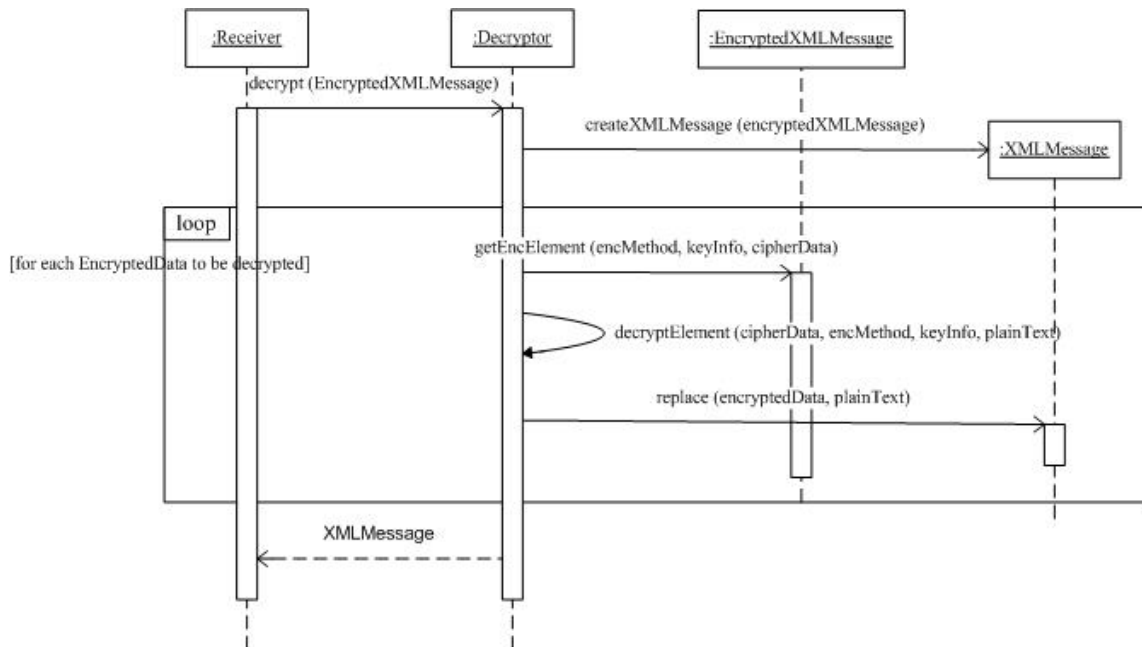
Alternate Flows:

If the key used in step d) is not the same as the one used in the encryption, then the decryption process fails.

Postcondition: The message has been decrypted.

Figure 10

Sequence Diagram for decrypting XML Elements



4.3.6 Implementation

- The designer should choose strong encryption algorithms to prevent attackers from breaking them such as Advanced Encryption Standard (AES) and DES (Data Encryption Standard) for symmetric encryption, and RSA (Rivest, Shamir, and Adleman) for asymmetric encryption.
- Asymmetric encryption or public-key encryption is more computationally intensive than symmetric encryption. However, symmetric encryption requires that both sender and receiver share a common key. A better practice will be to use the asymmetric encryption in combination with the symmetric encryption. Use symmetric encryption for the data and asymmetric encryption for secure key distribution.
- XML Encryption supports both symmetric and asymmetric encryption. This provides application flexibility; for example, a session uses symmetric encryption and key distribution uses asymmetric encryption.
- The following example illustrates how an encrypted part is embedded within an XML message.

Suppose you want to send a purchase order to the distribution office. This document contains details of the order such as what item to buy, quantity, and credit card information for payment. We want to keep the XML document simple just to focus on the encryption part.

```
<Order>
```

```
  <Item> Item X </Item>
```

```
<Quantity> 24 </Quantity>
<Payment Info>
  <Credit Card>
    <Number>1234566 </Number>
    <Expiration Date> 12/12/2010</Expiration Date>
  </Credit Card>
</Payment Info>
</Order>
```

Because Payment Info contains sensitive information, we want only to encrypt this element, so it can only be understood by the intended receiver.

```
<Order>
  <Item> Item X </Item>
  <Quantity> 24 </Quantity>
  <Encrypted Data>
    <Encryption Method Algorithm="AlgorithmX"/>
    <Cipher Data>
      <Cipher Value>ijutfrewsvbnmlkk </Cipher Value>
    </Cipher Data>
    <Key Info>
      <Key Name> KeyA </KeyName>
    </Key Info>
```

```
</Encrypted Data>
</Order>
```

The Payment Info element is replaced by the Encrypted Data element that includes all the information needed by the receiver. The Encryption Method element includes the algorithm used for the encryption. The Cipher Value contains the actual encrypted data. For this example, the Key Info element includes the name to identify the key.

4.3.6 Known Uses

Several vendors have developed tools that support XML Encryption:

- Xtradyne's WebService Domain Boundary Controller (WS-DBC) [Xtr]. The WS-DBC is an XML firewall that provides protection against malformed messages and malicious content, XML encryption, XML signature, and authentication, authorization, and audit.
- IBM - DataPower XML Security Gateway XS40 [IBM] parses, filters, validates schema, decrypts, verifies signatures, signs, and encrypts XML message flows.
- Forum Systems - Forum Sentry SOA Gateway [For] conforms to XML Digital Signature, XML Encryption, WS-Trust, WS-Policy and other standards.
- Microsoft .NET [Mic] includes APIs that support the encryption and decryption of XML data.

4.3.7 Consequences

This pattern presents the following advantages:

- Only users that know the key can decrypt and read the message. Each recipient can only decrypt parts of a message that are intended for him but is unable to decrypt the rest.
- The EncryptedData is an XML element that replaces the data to be encrypted. The EncryptedData as well as the EncryptedKey are composed by other subelements such as encryption method, key information, and cipher value.
- The entire XML message or only some parts can be encrypted.
- If both the sender and the receiver have not exchanged the keys previously, the key can be sent in the message encrypted using public key system.

The pattern also has some (possible) liabilities:

- The general liabilities of symmetric and asymmetric encryption still apply.
- The structure is rather complex and users may get confused.
- Unencrypted portions in the message, they may help a possible attacker. This might be improved by superencryption of the whole message at a lower level, e.g. using TLS.

4.3.8 Related Patterns

- This pattern includes a specialization of the Symmetric Encryption Pattern.
- The WS-Security Pattern [Has09] is a standard for securing XML messages using XML signature, XML Encryption, and security tokens.
- The Strategy Pattern [Gam94] defines how to separate the implementation of related algorithms from the selection of one of them.

The following specifications are related to XML Signature, but they have not been developed as patterns.

- The XML Key Management Specification (XKMS) [W3C01] specifies the distribution and registration of public keys, and works together with XML Encryption.
- WS-SecurityPolicy [OAS07] standard describes how to express security policies such as what algorithms are supported by a web service or what parts of an incoming message need to be signed or encrypted.

4.4 Summary

We presented two patterns: Symmetric Encryption and XML Encryption, the latter a specialization and extension of the first one. We showed these two patterns together to make clearer the logic behind XML Encryption, a rather complex pattern.

5. SIGNATURE PATTERNS

Data security has become one of the most important concerns among us especially for organizations that need to protect their information against attackers. An important security risk is that information can be modified during its transmission. How do we prove that a message came from a specific user? Digital signature uses public-key cryptography to provide message authentication by proving that a message was sent indeed from the sender who claims to have sent it [dig, Sta06]. The sender encrypts the message using his private key to sign it. In this case, the signature has at least the same length as the message. However, this approach wastes bandwidth and time. Thus, we need to reduce the length to the message before signing it. This can be done producing a digest through hashing. When the receiver gets the signed message, he verifies the signature by decrypting it using the sender's public key, thus proving that the message was encrypted by the sender. Also, digital signatures provide message integrity by verifying whether a message was modified during its transmission. Digital signatures can also protect the integrity and verify the origin of a digital document, e.g. a certificate, or of programs. Digital signatures provide also non-repudiation, the sender cannot deny having sent the message he signed. In several countries, including the U.S., digital signatures have legal validity.

An emerging use of web services that exchanges XML messages also can be target of attacks. Some security standards have been developed to reduce security risks. XML Signature is one of the basic standards in securing web services. This standard is a joint effort between the World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF). XML Signature defines how to digitally sign an entire XML message, part of an XML message, or an external object. XML Signature also includes hashing, but the pattern name follows the name of the standard. Because of the nature of XML documents, we need to convert the documents into a canonical form before we apply digital signatures. Note that XML Signature solves the same problem as the Digital Signature with Hashing pattern but in a more specialized context.

In this section, we present here two patterns: XML Signature and Digital Signature with Hashing patterns. The XML Signature pattern, a specialization of the Digital Signature with Hashing, is used to secure XML messages. We assume the reader is a designer intending to use message authentication in her design or a user intending to sign documents and who have a basic knowledge of cryptography and UML. We provide a solution with sufficient detail so as it can be used as a guideline for design of signature systems and for users of signed documents.

5.1 Digital Signature with Hashing

5.1.1 Intent

Digital Signature with Hashing allows a principal to prove that a message was originated from it. It also provides message integrity by indicating whether a message was altered during transmission.

5.1.2 Example

Alice in the Sales department wants to send a product order to Bob in the production department. The product order does not contain sensitive data such as credit card number, so it is not important to keep it secret. However, Bob wants to be certain that the message was created by Alice so he can charge the order to her account. Also, because this order includes the quantity of items to be produced, an unauthorized modification to the order will make Bob manufacture the wrong quantity of items. Eve can intercept the messages and may want to do this kind of modification.

5.1.3 Context

Participants of electronic transactions that need to exchange documents or messages through insecure networks and need to prove their origin and integrity. Stored legal documents need to be kept without modification and indicating their origin. Software sent by a vendor through the Internet requires to prove their origin.

We assume that a principal possesses a key pair: a private key that is secretly kept by the principal and a public key that is in a publicly-accessible repository. We assume that there is a mechanism for the generation of these key pairs and for the distribution of public keys.

5.1.4 Problem

In many applications we need to verify the origin of a message (message authentication). Since an impostor may assume the identity of a principal, how do we verify that a message came from a particular principal? Also, messages that travel through insecure channels can be captured and modified by attackers. How do we know that the message/document that we are receiving has not been modified?

The solution for these problems is affected by the following forces:

- For legal or business reasons we need to be able to verify who sent a particular message. Otherwise, we may not be sure of its origin and the sender may deny having sent it (repudiation). We assume the sender has signed the message to prove she is its author.
- Messages may be altered during transmission, so we need to verify that the data is in its original form when it reaches its destination.
- The length of the signed message should not be significantly larger than the original message; otherwise we would waste time and bandwidth.
- Producing a signed message should not require a large computational power or take a long time.

5.1.5 Solution

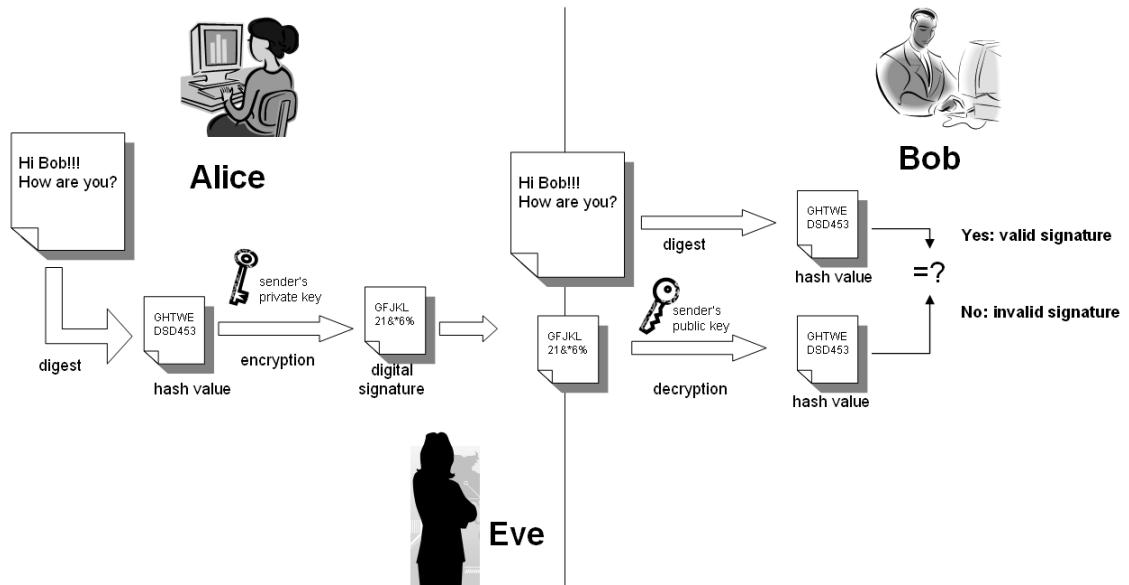
Apply properties of public key cryptographic algorithms to messages in order to create a signature that will be unique for each sender. The message is first compressed (hashed) to a smaller size (digest), and then it is encrypted using the sender's private key. When the signed message arrives at its target, the receiver verifies the signature using the sender's

public key to decrypt the message, if it produces a readable message, it could only have been sent by this sender. The receiver then generates the hashed digest of the received message and compares it to the received hashed digest, if it matches the message has not been altered.

This approach uses public key cryptography where one key is used for encryption and the other key for decryption. For digital signatures (SIG), we encrypt (E) the hash value of a message (H(M)) using the sender's private key (PrK): $SIG = E_{PrK}(H(M))$

We recover the hash value of the message (H(M)) by decrypting (D) the signature (SIG) using the sender's public key (PuK). If this produces a legible message, we can be confident that the sender created the message. Finally, we calculate the hash value of the message as $H(M) = D_{PuK}(SIG)$. If this value is the same as the message digest obtained when the signature was decrypted, then we know that the message has not been modified.

It is clear that the sender and receiver should both use the same encryption and hashing algorithms.



Structure

Figure 8 describes the class diagram for the Digital Signature Pattern.

A **Principal** may be a process, a user, or an organization that is responsible for sending or receiving messages. This **Principal** may have the roles of **Sender** or **Receiver**. A Sender may send a plain Message and/or a SignedMessage to a receiver.

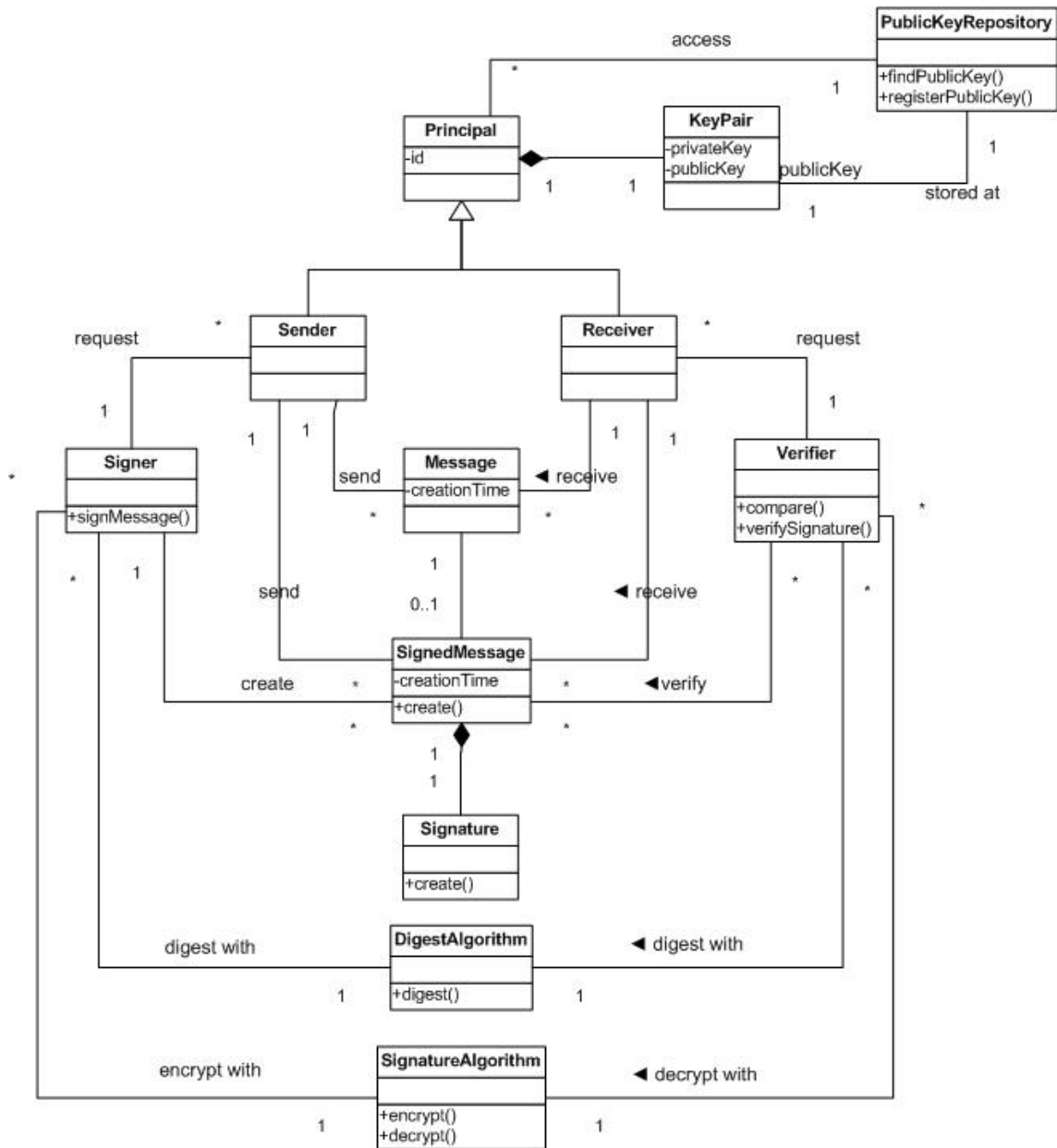
The **KeyPair** entity contains two keys: public and private, that belong to a **Principal**. The public key is registered and accessed through a repository, and the private key is kept secret by the owner. In a Public Key system, one key is normally used for encryption, while the other is used for decryption. **PublicKeyRepository** is a repository that contains public keys that can be available to anyone. The PublicKeyRepository may be located in the same local network as the principal or in an external network.

The **Signer** creates the **SignedMessage** that includes the **Signature** for a specific message. On the other side, the **Verifier** checks that the **Signature** within the **SignedMessage** corresponds to that message.

The Signer and Verifier use the **DigestAlgorithm** and **SignatureAlgorithm** to create and verify a signature respectively. The **DigestAlgorithm** is a hash function that condenses a message to a fixed length called a hash value or message digest. The **SignatureAlgorithm** encrypts and decrypts messages using public/private key pairs.

Figure 11

Class Diagram for Digital Signature Pattern



Dynamics

We describe the dynamic aspects of the Digital Signature Pattern using sequence diagrams for the use cases sign a message and verify a signature.

Sign a message (Figure 9):

Summary: A Sender wants to sign a message before sending it

Actors: A Sender

Precondition: A Sender has a public/private pair key

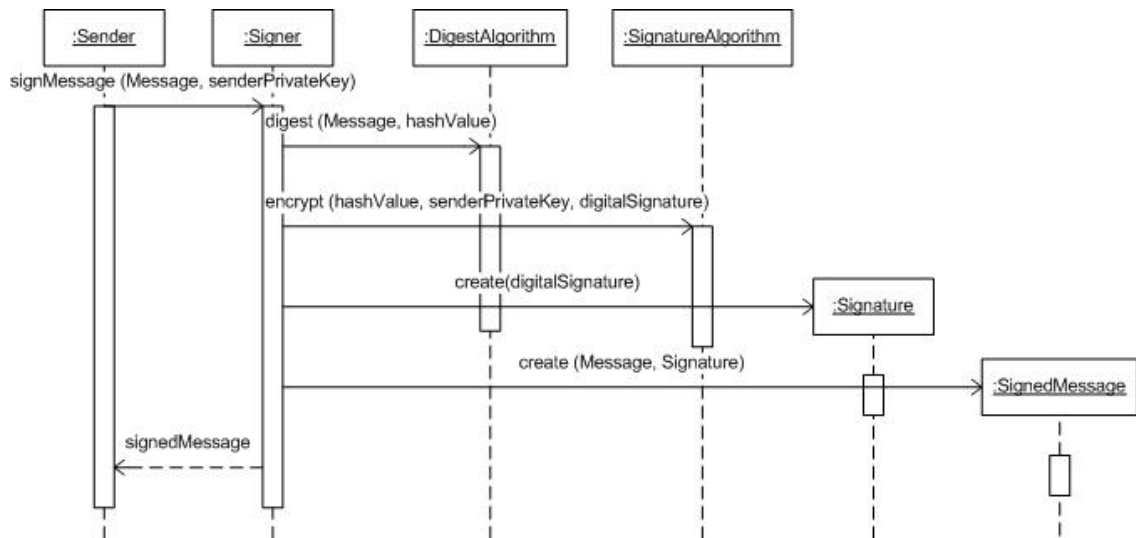
Description:

- g) A Sender sends the message and its private key to the signer.
- h) The Signer calculates the hash value of the message (digest) and returns it to the Signer.
- i) The Signer encrypts the hash value using the sender's private key with the Signature Algorithm. The output of this calculation is the digital signature value.
- j) The Signer creates the Signature object that contains the digital signature value.
- k) The Signer creates the SignedMessage that contains the original message and the Signature.

Postcondition: A SignedMessage object has been created.

Figure 12

Sequence Diagram for signing a message



Verify a Signature (Figure 10):

Summary: A receiver wants to verify that the signature corresponds to the received message.

Actors: A Receiver

Precondition: None

Description:

- i) A Receiver retrieves the sender's public key from the repository.
- j) A Receiver sends the signed message and the sender's public key to the verifier.
- k) The verifier decrypts the signature using the sender's public key with the Signature Algorithm.
- l) The verifier calculates the digest value of the message.
- m) The verifier compares the outputs from step c) and d).

n) The verifier sends an acknowledgement to the receiver that the signature is valid.

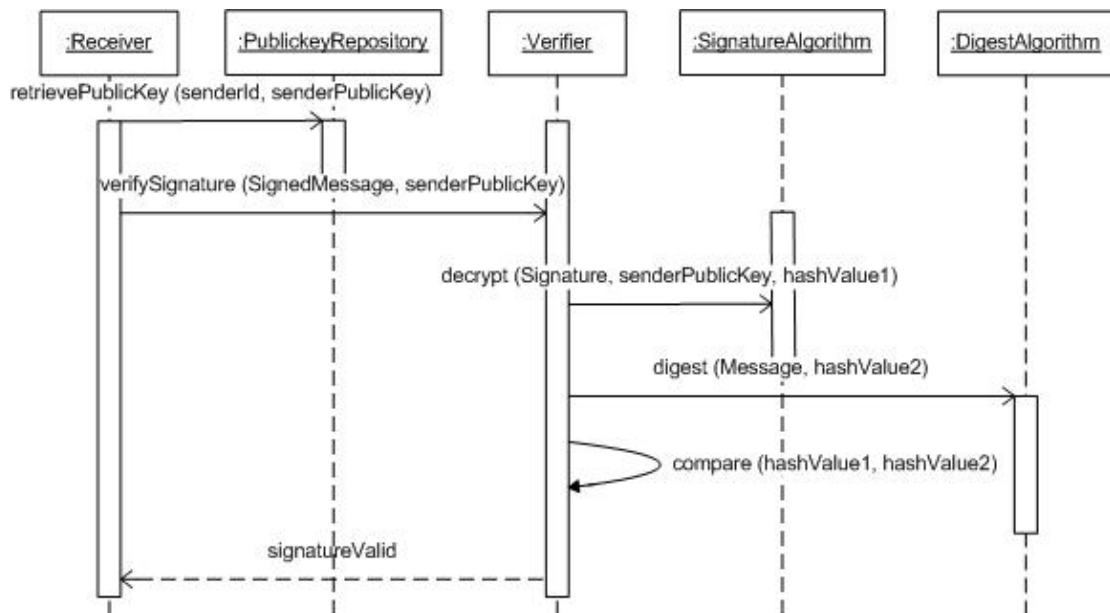
Alternate Flows:

- The outputs from step c) and d) are not the same. Then, the verifier sends an acknowledgement to the receiver that the signature failed.

Postcondition: The signature has been verified.

Figure 13

Sequence Diagram for verifying a signature



5.1.6 Implementation

- Use the Strategy Pattern [Gam94] to select different hashing and signature algorithms. The most widely used hashing algorithms are MD5 and SHA1. Those and others are discussed in [Sta06].

- A good hashing algorithm produces digests that are very unlikely produced by other meaningful messages, meaning that it is very hard for an attacker to create an altered message with the same hash value. The message digest should be encrypted after being signed to avoid man-in-the-middle attacks, where a person who captures a message could reconstruct its hash value.
- Two popular digital signature algorithms are RSA [RSA], and Digital Signature Algorithm (DSA) [Fed00, Sta06].
- The designer should choose strong and proven algorithms to prevent attackers from breaking them. The cryptographic protocol aspects, e.g. key generation, are as important as the algorithms used.
- The sender and receiver should have a way to agree on the hash and encryption algorithms used for a specific set of messages. XML documents indicate which algorithms they use and pre-agreements are not necessary.
- Access to the sender's public key should be available from a public directory or from certificates presented by the signer.
- Digital signatures can be implemented in different applications such as in email communication, distribution of documents over the Internet, or web services. For example, one can sign email's contents or any other document's content such as PDF. In both cases, the signature is appended to the email or document. When digital signatures are applied in web services, they are also embedded within XML messages. However, these signatures are treated as XML elements, and they have additional features such as signing parts of a message or external resources which can be XML or any other data type.

- When certificates are used to provide the sender's public key, there must be a convenient way to verify that the certificate is still valid [SOA01].
- There should be a way to authenticate the signer software [dig]. An attacker who gains control of a user's computer could replace the signing software with his own software.

5.1.7 Known Uses

Digital Signatures have been widely used in different products.

- Adobe Reader and Acrobat [Ado05] have an extended security feature that allows users to digitally sign PDF documents.
- CoSign [Arx] digitally signs different types of documents, files, forms, and other electronic transactions.
- GNUPG [Gnu] digitally signs e-mail messages.
- Java Cryptographic Architecture [Sunb] includes APIs for digital signature.
- Microsoft .Net [Mic07] includes APIs for asymmetric cryptography such as digital signature.
- XML Signature [W3C08] is one of the foundation web services security standards that defines the structure and process of digital signatures in XML messages.

5.1.8 Consequences

This pattern presents the following advantages:

- A principal's private key is used to sign the message. The signature is validated using its public key, which proves that the sender created and sent the message.

- When a signature is validated using a principal's public key, the sender cannot deny that he created and sent the message. If a message is signed using another private key that does not belong to the sender, the validity of the signature fails.
- If the proper precautions are followed (See 2.6), any change in the original message will produce a digest value that will be different from the value obtained after decrypting the signature using the sender's public key.
- A message is compressed into a fixed length string using the hash algorithm before it is signed. As a result, the process of signing is faster, and the signed message is much shorter.
- The available algorithms that can be used for digital signatures do not require very large amounts of computational power and do not take large amounts of time.

The pattern also has some (possible) liabilities:

- We need a well established Public Key Infrastructure that can provide reliable public keys. Certificates issued by some certification authority are the most common way to obtain this [Sta06].
- Both the sender and the receiver have to previously agree what signature and hashing algorithms they support. This is not necessary in XML documents because they are self-describing.
- Cryptographic algorithms create some overhead (time, memory, computational power), which can be reduced but not eliminated.
- Users must implement properly the signature protocol.

- There may be attacks against specific algorithms or implementations [dig]. These are difficult to use against careful implementations.
- This solution only allows one signer for the whole message. A variant or specialization, such as the XML Signature pattern, allows multiple signers.
- Digital signatures do not provide message authentication and replay attacks are possible [SOA01]. Nonces or time stamps could prevent this type of attacks.

5.1.9 Example Resolved

Alice and Bob agree on the use of a digital signature algorithm, and Bob has access to Alice's public key. Alice can then send a signed message to Bob. When the message is received by Bob, he verifies whether the signature is valid using Alice's public key and the agreed signature algorithm. If the signature is valid, Bob can be confident that the message was created by Alice. If the hash value is correct Bob also knows that Eve has not been able to modify the message.

5.1.10 Related Patterns

- Encryption/Decryption using public key cryptography [Bra98]
- Generation and Distribution of public keys [Leh02]
- Certificates [Mor06] are issued by a Certificate Authority (CA) that digitally signs them using its private key. A certificate carries a user's public key and allows anyone who has access to the CA's public key to verify that the certificate was signed by the CA.

- Strategy Pattern [Gam94], defines how to separate the implementation of related algorithms from the selection of one of them.

5.2 XML Signature

5.2.1 Intent

XML Signature allows a principal to prove that a message was originated from it. It also provides message integrity by defining whether a message was altered during transmission. The XML Signature standard [W3C08] describes the syntax and the process of generating and validating digital signatures for authenticating XML documents. XML Signature also provides message integrity. It requires canonicalization before hashing and signing.

5.2.2 Example

Alice in the Sales department wants to send product orders to Bob in the production department. The product orders are XML documents and do not contain sensitive data such as credit card number, so it is not important to keep them secret. Each order must be signed by Alice's supervisor Susie to indicate approval. Bob wants to be certain that the message was created by Alice so he can charge the order to her account and also needs to know that the orders are approved. Because the orders include the quantity of items to be produced, an unauthorized modification to an order will make Bob manufacture the wrong quantity of items. Eve can intercept the messages and may want to do this kind of modification.

5.2.3 Context

Users of web services send and receive SOAP messages through insecure networks such as the Internet and need to prove their origin and integrity. During their transmission these messages can be subject to a variety of attacks.

We assume that a principal possesses a key pair: a private key that is secretly kept by the principal and a public key that is in a publicly-accessible repository. We assume that there is a mechanism for the generation of these key pairs and for the distribution of public keys.

5.2.4 Problem

In many applications we need to verify the origin of a message (message authentication). Since an impostor may assume the identity of a principal, how do we verify that a message came from a particular principal? Also, messages that travel through insecure channels can be captured and modified by attackers. How do we know that the message/document that we are receiving has not been modified?

- For legal or business reasons we need to be able to verify who sent a particular message. Otherwise, we may not be sure of its origin and the sender may deny having sent it (repudiation). We assume the sender has signed the message to prove she is its author.
- Messages may be altered during transmission, so we need to verify that the data is in its original form when it reaches its destination.

- The length of the signed message should not be significantly larger than the original message; otherwise we would waste time and bandwidth.
- Producing a signed message should not require a large computational power or take a long time.
- We need to express a digital signature in a standardized XML format, so interoperability can be ensured between applications.
- There may be situations where we want to ensure proper origin or integrity in specific parts of a message. For example, an XML message can travel through many intermediaries that add or subtract information, so if we sign the entire message, the signature would have no meaning. Thus, we should be able to sign portions of a message.

5.2.5 Solution

Apply cryptographic algorithms to messages in order to create a signature that will be unique for each message. First, the data to be signed may need to be transformed before applying any digest algorithm. The series of XML elements (that includes other subelements) is canonicalized before applying a signature algorithm. Canonicalization is a type of transform algorithm that converts data into a standard format, to remove differences due to layout formatting. This process is required because XML is a flexible language where a document can be represented in different ways that are semantically equal. Thus, after calculating the canonical form, both the sender and the receiver will sign and verify the same XML data respectively. After applying a canonicalization

algorithm, the result value is digested and then encrypted using the sender's private key. Finally, the signature, in XML form, is embedded in the message.

In the other side, the receiver verifies the signature appended in the signed message. The verification process has two parts: reference verification and signature verification. In the reference verification, the verifier recalculates the digest value of the original data. This value is compared with the digest value included in the signature. If there is any mismatch, the verification fails. In the signature verification, the verifier calculates the canonical form of the signed XML element, and then applies the digest algorithm. This digest value is compared against the decrypted value of the signature. The decryption is done using the sender's public key.

There are three types of XML Signature: enveloped, enveloping and detached signature. In an enveloped signature, the signature is a child element of the signed data. For example, when you sign the entire XML message, the signature is embedded within the message. An enveloping signature is a signature where the signed data is a child of the signature. You can sign elements of a signature such as the Object or KeyInfo element. A detached signature is calculated over external network resources or over elements within the message. In the latter case, the signature is neither an enveloped nor an enveloping signature.

Structure

Figure 11 describes the structure of the XML Signature Pattern. Note that the upper part of this figure is almost the same as Figure 1.

A **Principal** may be a process, a system, a user, or an organization that sends and receives **XMLMessages** and/or **SignedXmlMessages**. This principal may have the roles of **Sender** and **Receiver**.

Both an **XMLMessage** and a **SignedXMLMessage** are composed by XML elements, but this is only shown in the **SignedXMLMessage**. Each **XMLElement** may be a **SingleElement** that does not have any children or be a **Composite** element which is composed by other XML elements.

The **XMLSigner** and the **XMLVerifier** create and verify a **Signature**, respectively. A **Signature** element is an XML element that has two required children: **SignedInfo** and **SignatureValue** and two optional children: **KeyInfo** and **Object**.

The **SignedInfo** element is the one that is actually signed. It contains one or more **Reference** elements, the canonicalization algorithm identifier, and the signature algorithm identifier. The Canonicalization algorithm is used to convert the **SignedInfo** element into a standard form before it is signed or verified. The Signature algorithm includes also a digest algorithm that is applied after calculating the canonical form of the **Signed Info** in both process creation and verification of XML signatures.

Each **Reference** element includes a Uniform Resource Identifier (URI), a hash value (DigestValue), the digest algorithm identifier (DigestMethod), and an optional list of **Transform** elements. The URI is a pointer that identifies the data to be signed. It can point to an element inside an XML message, an element inside the Signature element such as Object or KeyInfo, or resources located in the Internet. The DigestValue contains a hash value after applying the digest algorithm to the data pointed by its URI. If the Transform element exists, it includes an ordered list of transform algorithms that are applied to the data before being digested.

The **SignatureValue** element includes the value of the digital signature.

If the **KeyInfo** is present, it indicates the information about the sender's public key that will be used to verify the signature. This flexible element may contain certificates, key names, and other public keys forms. Additional information about this element can be found in [W3C08].

The optional **Object** element may contain **SignatureProperties** and/or a **Manifest**. The **SignatureProperty** identifies properties of the signature itself such as the date/time when the signature was created. The **Manifest** element includes one or more Reference elements same as the **Reference** element within the SignedInfo. They are semantically equal; however, each Reference in the SignedInfo has to be validated in order to consider a valid signature. On the other hand, the list of Reference elements within the Manifest is validated.

The sender and receiver must use the same hash, signature, and canonicalization algorithms. XML documents are self-descriptive and indicate this information so the sender only needs to find the corresponding algorithms.

Dynamics

We describe the dynamic aspects of the XML Signature Pattern using sequence diagrams for the use cases sign different XML elements of an XML message and verify an XML signature with multiple references.

Sign an XML message (Figure 12):

Summary: A sender wants to sign specified XML elements of an XML message.

Actors: A sender

Precondition: A sender has a private/public key pair.

Description:

- a) A sender requests the signer to sign different XML elements of a message.
- b) The signer calculates the digest value over the XML element.
- c) The signer creates the <Reference> element including the digest value and using the digest algorithm.
- d) Repeat steps b) and c) for each XML element to be signed.
- e) The signer creates the <SignedInfo> that includes the Reference elements, the canonicalization algorithm identifier, and the signature algorithm identifier.
- f) The signer applies the canonicalization algorithm to the <SignedInfo> element.

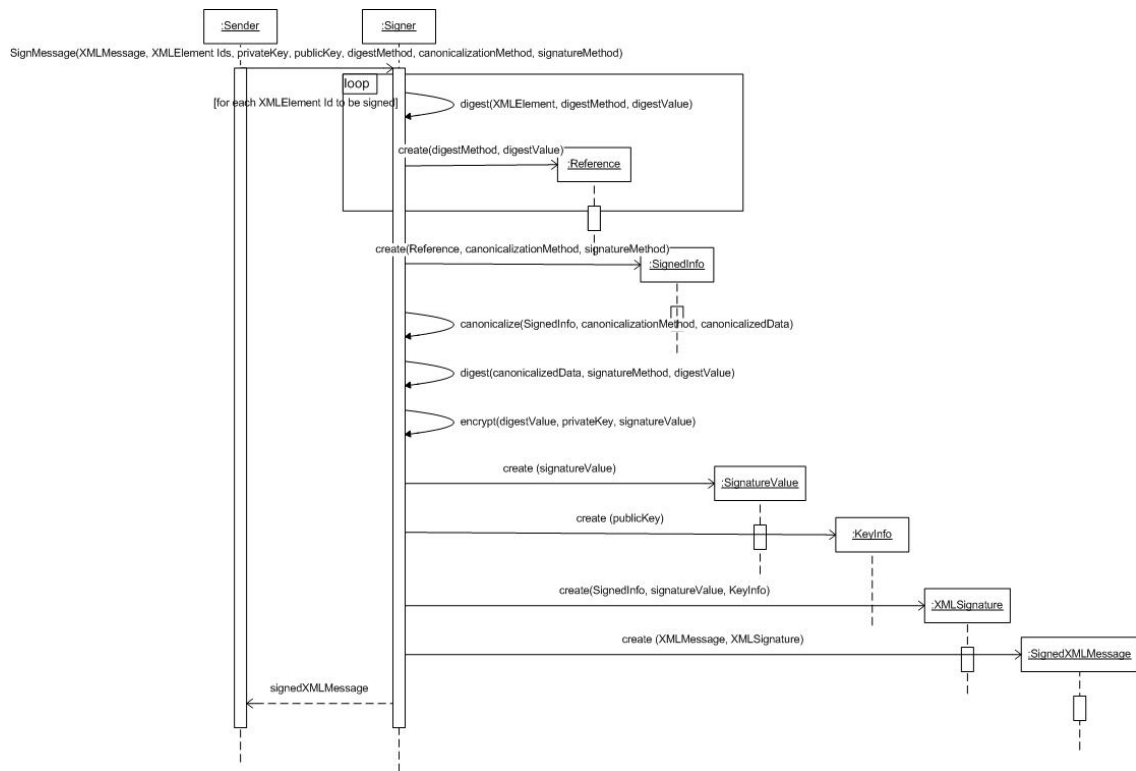
- g) The signer signs the output from step f). First, it applies the digest algorithm, and then it encrypts the digest using the sender's public key. The output is the signature value.
- h) The signer creates the <SignatureValue> element that includes the signature value.
- i) The signer created the <KeyInfo> element that holds the sender's public key that will be used to verify the signature.
- j) The signer creates the <Signature> element that includes the <SignedInfo>, the <SignatureValue>, and the <KeyInfo> elements.
- k) The signer creates the SignedXMLMessage that includes the Signature and the XMLMessage.

Alternate Flows: None

Postcondition: The specified elements of the document have been signed

Figure 15

Sequence Diagram for signing an XML message



Verify an XML signature with multiple references (Figure 13):

Summary: A receiver wants to verify the signature of a received document.

Actors: A Receiver

Precondition: None

Description:

- h) A receiver requests to verify the signature that is included in the SignedXMLMessage.

- i) The verifier obtains the signature elements such as the <SignedInfo> which includes the <Reference> elements, the <SignatureValue>, and the <KeyInfo> elements.
- j) The verifier calculates the digest value over the XML element that is pointed (URI) in the <Reference> element using the digest algorithm specified in the <Reference> element as well.
- k) The verifier compares the output from step c) against the digest value specified in the Reference element.
- l) Repeat step c) and d) for each <Reference> included in the <SignedInfo> element.
- m) The verifier canonicalizes the <SignedInfo> element using the canonicalization method specified in the <SignedInfo>.
- n) The verifier digests the output from step f) using the digest algorithm specified in the Signature Algorithm.
- o) The verifier decrypts the signature value using the sender's public key (<KeyInfo>).
- p) The verifier compares the outputs from step f) and h).
- q) The verifier sends an acknowledgement to the receiver that the signature is valid.

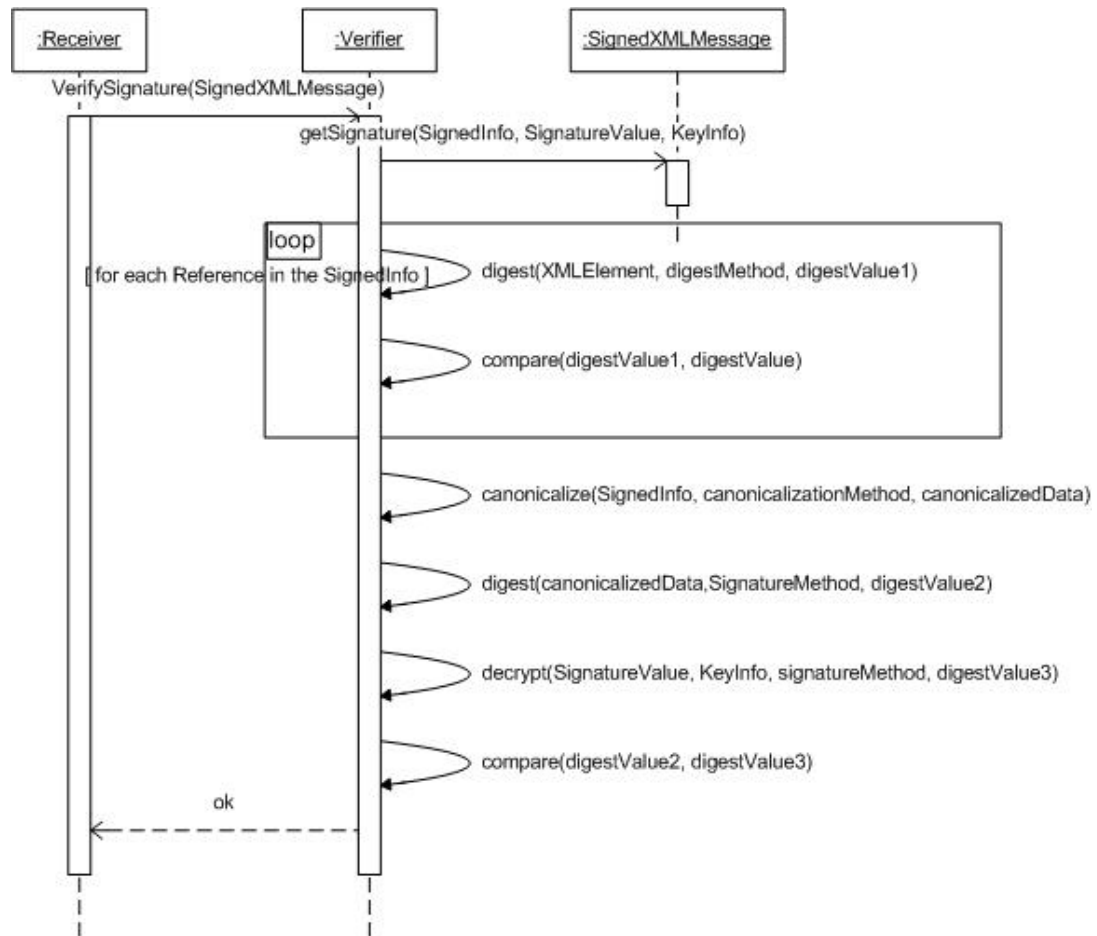
Alternate Flows:

- If the values compared in step d) are not the same, then the signature is invalid.
- If the outputs in the step i) are not the same, then the validation fails.

Postcondition: The signature is validated.

Figure 16

Sequence Diagram for verifying an XML signature



5.2.6 Implementation

- Identifiers of algorithms used to create a signature are attached along with the signature, so they also should be protected from being modified by attackers.
- XML documents may be parsed by different processors, and also XML allows some flexibility without changing the semantic of the message. Thus, we need to convert the data to be signed to a standard format.

- All the signers of a given document should have the same level of trust to avoid misleading the receivers about the trust of the whole message. Allowing untrusted signers might give them a better chance to attack the message.
- Use the Strategy Pattern [Gam94] to select different hashing and signature algorithms. The most widely used hashing algorithms are MD5 and SHA1. Two popular digital signature algorithms are RSA [RSA] and Digital Signature Algorithm (DSA) [Fed00].
- If needed the data to be signed needs to be transformed using transformation algorithms before producing a digest. For instance, if the object to be signed is an image, it needs to be converted into text.
- It is recommendable the use of certificates issued by an Certification Authority that are trusted by the sender and the receiver.

5.2.7 Known Uses

Several vendors have developed tools that support XML Signature.

- IBM - DataPower XML Security Gateway XS40 [IBM] parses, filters, validates schema, decrypts, verifies signatures, signs, and encrypts XML message flows.
- Xtradyne – Xtradyne’s WS-DBC [Xtr]. The Web Services Domain Boundary Controller is an XML firewall that provides protection against malformed messages and malicious content, XML encryption, XML signature, and authentication, authorization, and audit.
- Forum Systems - Forum Sentry SOA Gateway [For] conforms to XML Digital Signature, XML Encryption, WS-Trust, WS-Policy and other standards.

- Microsoft .NET [Mic] includes API that support the creation and verification of XML digital signatures.
- Java XML Digital Signature API [Mul07] allows to generate and validate XML signatures

5.2.8 Consequences

This pattern presents the following advantages:

- A principal's private key is used to sign the message. The signature is validated using its public key, which proves that the sender created and sent the message.
- When a signature is validated using a principal's public key, the sender cannot deny that he created and sent the message. If a message is signed using another private key that does not belong to the sender, the validity of the signature fails.
- Any change in the original message will produce a digest value that will be different from the value obtained after decrypting the signature using the sender's public key.
- Before applying any signature algorithm, the data is compressed to a short fixed-length string. In XML Signature, digest algorithms are used two times; one is used to digest data to be signed indirectly, and the other digest algorithm is used to digest the canonical form of the SignedInfo element.
- Any change in the data that was indirectly signed will produce another digest that will invalidate the signature.

- The available algorithms that can be used for digital signatures do not require very large amounts of computational power and do not take large amounts of time.
- An XML signature is an XML element that is embedded in the message. The XML signature is composed of several XML elements that include information such as the value of the signature, the key that will be used to verify the signature, and algorithms used to compute the signature. This standard format helps XML parsers to better understand signature elements during the validation process.
- This pattern supports also message authentication code (MAC). Both signatures and MACs are syntactically identical. The difference between them is that signatures use public key cryptography while MAC uses a shared common key.
- The data being signed is pointed by its URI (Uniform Resource Identifier), so elements within XML messages and external network resources can be located using their identifiers.
- The SignedInfo is the element that is actually signed. It includes the references that point the data being signed along with their digest values, and algorithms identifiers. Thus, the XML signature also protects the algorithm identifiers from modification.
- XML Signature uses canonicalization algorithms to ensure that different representations of XML are transformed into a standard format before applying any signature algorithm.
- XML documents are self-describing and the sender and receiver don't need to agree in advance on the algorithms to be used.

The pattern also has some (possible) liabilities:

- We need a well established Public Key Infrastructure that can provide reliable public keys. Certificates issued by some certification authority are the most common way to obtain this [Sta06]. There is a public key standard for XML that should be used.
- Users must implement properly the signature protocol.
- There may be attacks against specific algorithms or implementations [dig]. These are difficult to use against careful implementations.
- Signing and verifying XML messages may create a significant overhead.
- The pattern does not describe the complete standard. For example, details of transforms and key values have been left out for simplicity [W3C08].

5.2.9 Example resolved

Alice and Susie sign each product order sent to Bob. Bob has access to Alice's and Susie's public keys. When the message is received by Bob, he verifies whether the signatures are valid using Alice's and Susie's public keys and the signature algorithm specified in the order. If the signature are valid, Bob can be confident that the message was created by Alice and approved by Susie. If the hash value is correct Bob also knows that Eve has not been able to modify the message.

5.2.10 Related Patterns

- This pattern is a specialization of the Digital Signature with Hashing Pattern.

- WS-Security Pattern [Has09] is a standard for securing XML messages using XML signature, XML Encryption, and security tokens.

The following specifications are related to XML Signature, but they have not been expressed as patterns.

- The XML Key Management Specification (XKMS) [W3C01] specifies the distribution and registration of public keys, which works together with the XML Signature.
- WS-SecurityPolicy [OAS07] standard describes how to express security policies such as what algorithms are supported by a web service or what parts of an incoming message need to be signed or encrypted.

5.3 Summary

We presented two patterns: Digital Signature with Hashing and XML Signature, the latter a specialization of the first one for a more specific context. Since the XML pattern solves the same problem it repeats the general aspects of the Digital Signature pattern but repeating this information allows the XML pattern to be used alone. We showed these two patterns together to make clearer the logic behind XML Signature, a rather complex pattern.

6. WS-SECURITY PATTERN

WS-security was originally developed by IBM, Microsoft, VeriSign, and Forum Sentry, and it was approved as an OASIS standard on July 1, 2006. WS-Security defines how to secure web services by providing message integrity, message confidentiality, and message authentication. It describes how to attach XML encryption and XML signature within SOAP messages. Also, it defines how to embed security tokens e.g. X.509 certificates and Kerberos to SOAP messages. WS-Security is a flexible standard that support multiple security tokens, multiple encryption technologies, and multiple signature formats.

WS-Security is an OASIS standard that describes how SOAP messages can be secured through message integrity, message authentication, and message confidentiality. WS-Security is a flexible protocol that supports different formats of security tokens, different encryption technologies, and different signature formats. WS-Security does not define new security mechanisms, but it leverages existing technologies such as XML Encryption, XML Signature, and Security Tokens e.g. Kerberos Tickets and X.509 certificates. We describe this standard in the forma of a pattern using a common template. We describe briefly its two supporting patterns: XML Encryption and XML Signature.

6.1 Intent

The WS-Security standard [OAS06] describes how to embed existing security mechanisms such as XML Encryption [W3C02], XML digital signature [W3C08], and security tokens into SOAP messages in order to provide message confidentiality, integrity, and authentication, as well as non-repudiation.

6.2 Context

Users of web services send and receive SOAP messages through insecure channels such as the Internet.

6.3 Problem

Sending message through insecure channels expose the messages to a variety of attacks, including illegal reading or modification, replay, and the sender can deny having sent a specific message [Sta06]. We have cryptographic solutions for these problems; however, there are many algorithms and protocols and we need to make a selection self-descriptive.

The solution for this problem is affected by the following forces:

- *Interoperability.* We need a common format in SOAP messages in order to add security features, so both senders and receivers can be able to process messages that contain security features without need for previous agreements.
- *Fine degree of protection.* SOAP messages may travel in a network environment through many intermediaries and different users may need access to different

parts of them. We may need to protect different parts of a message in different ways.

6.4 Solution

Define areas in the message format that specify parameters that specify security mechanisms such as encryption, digital signatures, and security tokens.

A SOAP message is composed of a body and an optional header. Three major elements can be embedded within the header of a message: XML Encryption, XML Signature, and security tokens. If an element within the message is signed, the header can include information about the signature such as the algorithm, the key, and the value of the signature. For XML Encryption, the security header can enclose a list of references that point to the parts of the message that have been encrypted and how.

Structure

Figure 14 describes the structure for WS-Security.

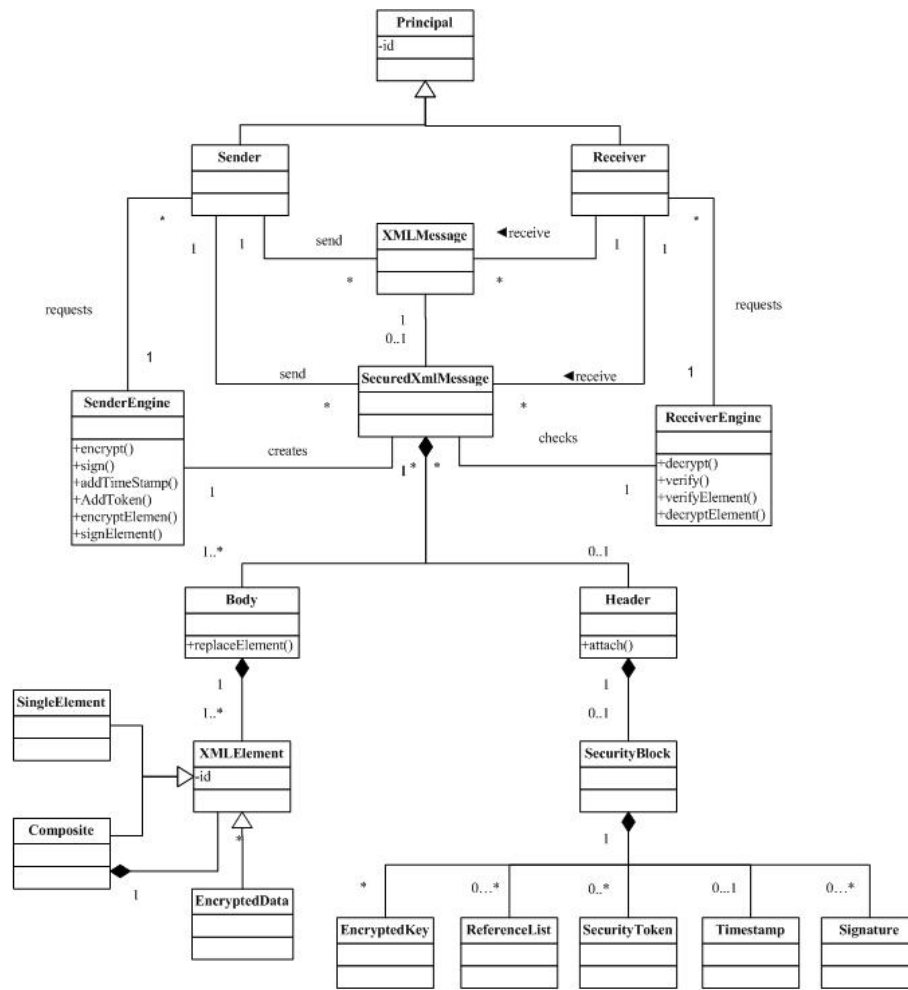
A **Principal** may be a system, a user, or an organization that sends and receives **XMLMessages**. This principal may have the roles of **Sender** and **Receiver**. The **SenderEngine** includes a Sender and an Encryptor, while the **ReceiverEngine** includes a Verifier and a Decryptor.

Security Tokens such as Username/Password, X.509 Certificates, and Kerberos Tickets are used for authentication and authorization purposes.

XMLMessages are composed of a **Body** and an optional **Header**. A **Header** may contain a **Security Block** which may enclose **Timestamp**, **EncryptedKey**, **ReferenceList**, **SignedElement**, and **SecurityToken** elements. Timestamps provide the time of creation and expiration of a message. **EncryptedKey** element represents the key used to encrypt parts or the entire message, and this key is encrypted according to XML Encryption standard. The **ReferenceList** element points to the parts of the message that are encrypted with XML Encryption. The **SignedElement** holds information about the signatures generated according to XML Signature standard. The **Body** is a collection of **Elements**, some of which are **Encrypted Data**. Elements can be structured into Composite hierarchies.

Figure 17

Class Diagram for WS-Security Pattern



Dynamics

We describe the dynamic aspects of the WS-Security Standard using sequence diagrams for the use cases: encrypt an element using a symmetric key that is itself encrypted using a security token and sign an element using a security token.

Encrypt an element using an encrypted key (Figure 15):

Summary: A Sender encrypts an element using a symmetric key that is itself encrypted using a security token.

Actors: A Sender

Precondition: The sender has a symmetric key for this communication.

Description:

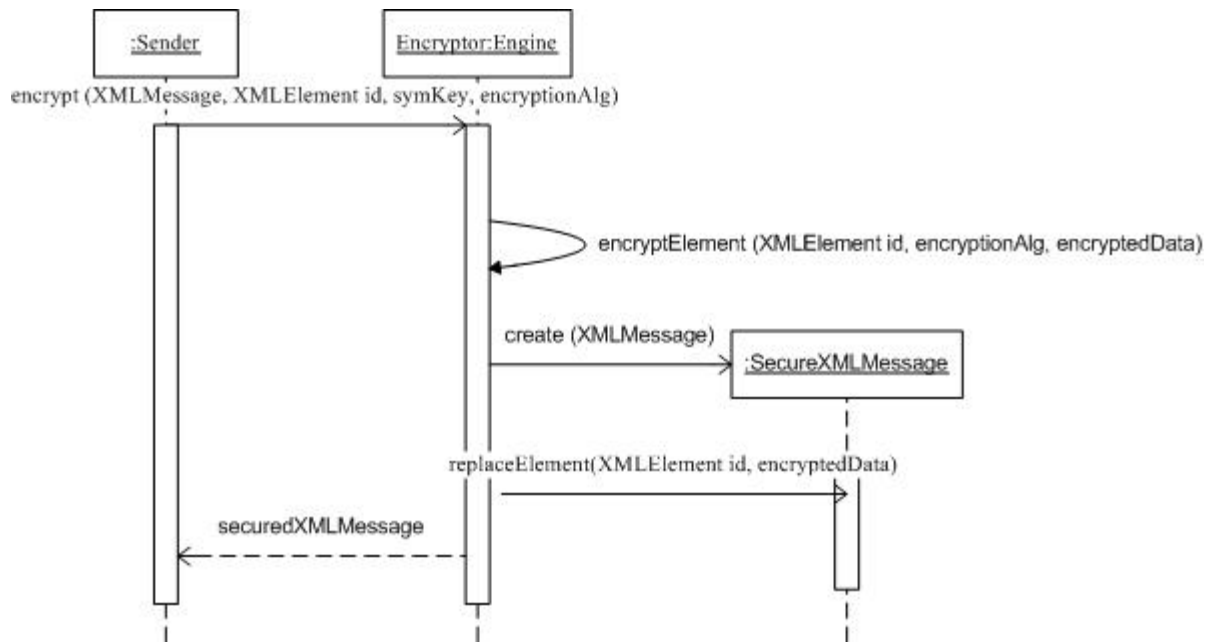
- a) A Sender requests to the Encryptor to encrypt an XML element
- b) The Encryptor encrypts the XML element using a symmetric key and the encryption method provided by the sender.
- c) The Encryptor creates the SecureXMLMessage that will contain the encrypted element.
- d) The Encryptor replaces the plain XML element with the output from step b).
- e) The Encryptor sends the secured XML Message to the sender, who can now send it to some Receiver.

Alternate Flows:

Postcondition: The encrypted element is attached to the message.

Figure 18

Sequence Diagram for encrypting a message



Sign an element (Figure16):

Summary: A Sender signs an element.

Actors: A Sender

Precondition: The sender has a private key in some PKI system.

Description:

- a) A Sender requests to the Signer to sign an XML element.
- b) The Signer signs the XML element using the sender's private key and the signature algorithm provided by the sender.
- c) The Signer created the Secured XML Message that will contain the digital signature.
- d) The Signer attaches the signature into the security block.

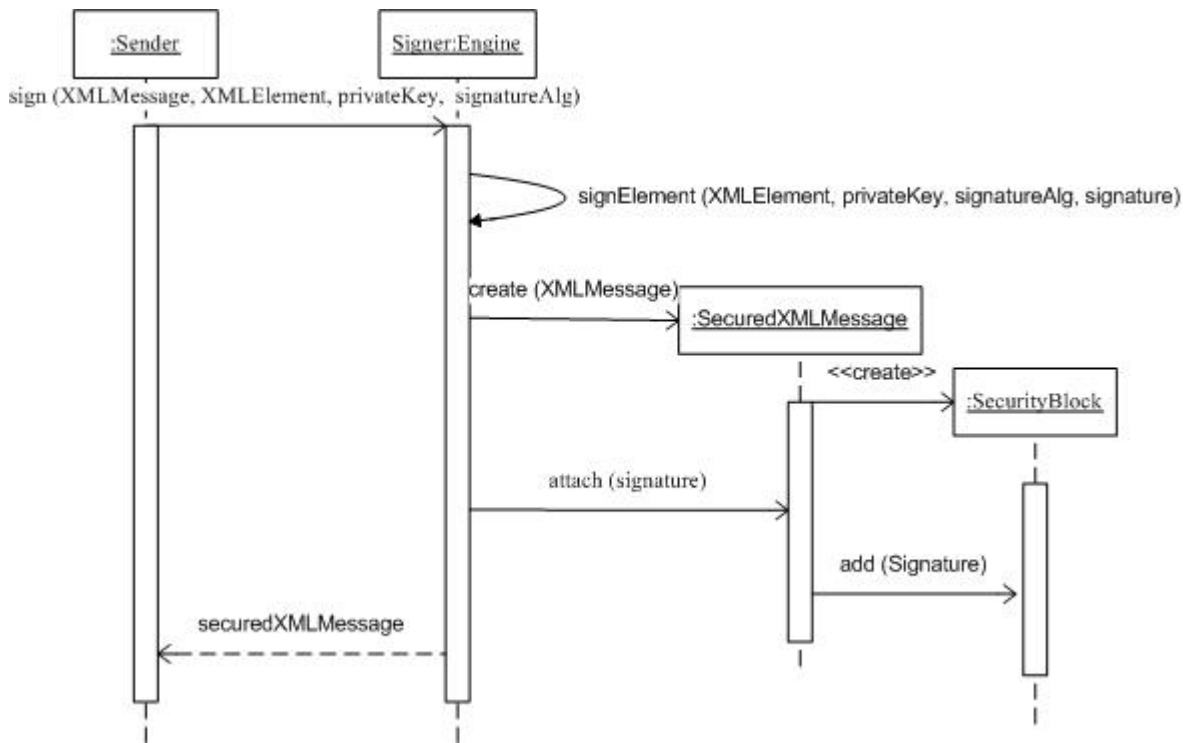
e) The Signer sends the secured XML message to the sender.

Alternate Flows:

Postcondition: The signature has been attached to the header.

Figure 19

Sequence Diagram for signing a XML element



6.5 Implementation

To implement WS-Security standard, the following tasks need to be done:

1. Clients need to have knowledge of cryptographic algorithms such as security token formats, signature formats and encryption technologies.

2. A message can have multiple headers if they are targeted for different recipients.

In other words, message security information targeted to different recipients must be in different headers.

6.6 Known Uses

Several vendors have developed products that support WS-Security.

- Xtradyne's WS-DBC (Web Service Domain Boundary Controller) [Xtr] is an XML firewall that supports the WS-Security standard and other standards.
- IONA Artix [Ion]
- Forum Systems - Forum Sentry SOA Gateway [For] conforms to XML Digital Signature, XML Encryption, WS-Trust, WS-Policy and other standards.

6.7 Consequences

This pattern presents the following advantages:

- Using the header of a SOAP message we can specify the security features of a message such as XML encryption, XML signatures, and security tokens.
- We can specify different parts of a message with different types of encryption, different keys, or different signatures.

The pattern also has some (possible) liabilities:

- This pattern does not describe details of encryption, digital signatures, or security tokens. Those require separate standards.

- WS-Security does not tell you whether you should sign or encrypt whole message, a part of it, or only the header. It is up to the designer to define these aspects.
- WS-Security is an immature specification which is still changing.

6.8 Related Patterns

- WS-Security uses the XML Digital Signature [Has09a] and XML Encryption [Has09b] patterns.
- Secure Channel is a way to transport messages providing message authentication, message confidentiality, and message integrity [Bra98].

6.9 Summary

WS-Security allows for a SOAP message to identify the sender, sign the message, and encrypt message contents. WS-Security does not invent new security mechanism but reuse existing specifications such as XML Encryption and XML Signature.

7. CONCLUSION AND FUTURE WORK

We have developed three patterns for web services security standards. However, we observed that these standards have many details that may confuse the readers. Thus, we developed also two abstract patterns in order to have a general idea how the protocols work. We wrote patterns for XML Signature [Has09a], XML Encryption [Has09b], and WS-Security [Has09c]. XML Encryption is a flexible standard that provides confidentiality by hiding all or parts of XML messages. Likewise, due to its flexibility, XML Signature also allows you to sign all or parts of XML messages. WS-Security is a protocol that provides message authenticity, confidentiality, and integrity by leveraging XML Encryption, XML Signature, and Security Tokens. While these are not all the cryptographic standards used in web services, they are the most important.

There is a large number of web services standards and it is hard for users and tool developers to find the right one. Thus, we need to develop more patterns for these standards, so we can compare them and understand them better [Fer06]. In order to provide a broad perspective we enumerated the current standards for web services, providing references to the complete standard [Fer09].

Future work will include completing our development of other web services security patterns such as WS-Trust, WS-Federation, WS-SecureConversations, XKMS (Key Management Specification), and WS-SecurityPolicy.

REFERENCES

- [Add06a] W3C, Web Services Addressing 1.0 – Core, 9 May 2006,
<http://www.w3.org/TR/ws-addr-core/>
- [Add06b] W3C, Web Services Addressing 1.0 – SOAP Binding, 9 May 2006,
<http://www.w3.org/TR/2006/REC-ws-addr-soap-20060509/>
- [Add06c] W3C, Web Services Addressing 1.0 – WSDL Binding, 29 May 2006,
<http://www.w3.org/TR/ws-addr-wsdl/>
- [Add06d] Web Services Addressing 1.0 – Metadata, 4 September 2007.
<http://www.w3.org/TR/ws-addr-metadata/>
- [Ado] Adobe System Incorporated, Digital Signatures,
<http://www.adobe.com/security/digsig.html>
- [Aro05] A. Arora et al., The WS-Management Catalog, June 2005,
http://www.dell.com/downloads/global/corporate/standards/ws_management_catalog.pdf
- [Ato07] OASIS, Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.1, 16 April 2007, <http://docs.oasis-open.org/ws-tx/wstx-wsat-1.1-spec-os/wstx-wsat-1.1-spec-os.html>
- [Arx] Arx, Digital Signature Solution (Standard Electronic Signatures),
<http://www.arx.com/products/cosign-digital-signatures.php>

- [Avdl04] OASIS, Application Vulnerability Description Language 1.0, May 2004, <http://www.oasis-open.org/committees/download.php/7145/AVDL%20Specification%20V1.pdf>
- [Bio03] OASIS, XML Common Biometric Format, August 2003, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xcbf
- [Bpel07] OASIS, Web Services Business Process Execution Language Version 2.0, 11 April 2007, <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>
- [Bra98] A. Braga, C. Rubira, and R. Dahab, “Tropyc: A pattern language for cryptographic object-oriented software”, Chapter 16 in *Pattern Languages of Program Design 4* (N. Harrison, B. Foote, and H. Rohnert, Eds.). Also in *Procs. Of PloP'98*, http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/
- [Bus07] OASIS, Web Services Business Activity (WS-BusinessActivity) Version 1.1, 12 July 2007, <http://docs.oasis-open.org/ws-tx/wstx-wsba-1.1-spec-errata-os.pdf>
- [Cer06] OASIS, Web Services Security: X.509 Certificate Token Profile 1.1, 1 February 2006, <http://www.oasis-open.org/committees/download.php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf>
- [Con07] OASIS, WS-SecureConversation 1.3, 1 March 2007, <http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512/ws-secureconversation-1.3-os.html>

- [Con07] OASIS, Web Services Context Specification (WS-Context) Version 1.0, 2 April 2007, <http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.html>
- [Coo07] OASIS, Web Services Coordination (WS-Coordination) Version 1.1, 12 July 2007, <http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-errata-os.pdf>
- [Cor05] W3C, Web Services Choreography Description Language Version 1.0, 9 November 2005, <http://www.w3.org/TR/ws-cdl-10/>
- [Del07] N. Delessy, E.B.Fernandez, and M.M. Larrondo-Petrie, "A pattern language for identity management", Proc. of the 2nd IEEE Int. Multiconference on Computing in the Global Information Technology (ICCGI 2007), March 4-9, Guadeloupe, French Caribbean.
- [dig] Digital signature, http://en.wikipedia.org/wiki/Digital_signature
- [Dis05] Microsoft Corporation, Inc., Web Services Dynamic Discovery (WS-Discovery), April 2005, <http://specs.xmlsoap.org/ws/2005/04/discovery/ws-discovery.pdf>
- [Dsi07] OASIS, Digital Signature Service Core Protocols, Elements, and Bindings Version 1.0, April 2007, <http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf>
- [ebX05a] ebXML Registry Services and Protocols Version 3.0, 2 May 2005, <http://www.oasis-open.org/specs/#ebxmlrsv3.0>
- [ebX05b] ebXML Registry Information Model Version 3.0, 2 May 2005, <http://www.oasis-open.org/specs/#ebxmlrsv3.0>

- [Enc02] W3C, XML Encryption Syntax and Processing, 10 December 2002,
<http://www.w3.org/TR/xmlenc-core/>
- [Enu06] W3C, Web Services Enumeration (WS-Enumeration), 15 March 2006,
<http://www.w3.org/Submission/WS-Enumeration/>
- [Eve06] W3C, Web Services Eventing (WS-Eventing), 15 March 2006,
<http://www.w3.org/Submission/WS-Eventing/>
- [Fed99] Federal Information Processing Standards Publication, “Data Encryption
Data (DES),” 25 October 1999, <http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf>
- [Fed00] Federal Information Processing Standard, “Digital Signature Standard,” 27
January 2000, [http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-
change1.pdf](http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf)
- [Fed01] Federal Information Processing Standards Publication, “Advanced
Encryption Standard,” 26 November 2001,
<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- [Fed03a] S. Bajaj et al, WS-Federation: Active Requestor Profile Version 1.0, 8 July
2003, [http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-
fedact/ws-fedact.pdf](http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fedact/ws-fedact.pdf)
- [Fed03b] S. Bajaj et al, WS-Federation: Passive Requestor Profile Version 1.0, 8 July
2003, [http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-
fedpass/ws-fedpass.pdf](http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fedpass/ws-fedpass.pdf)
- [Fed06] H. Lockhart et al, Web Services Federation Language (WS-Federation)
Version 1.1, December 2006,

http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-fed/WS-Federation-V1-1B.pdf?S_TACT=105AGX04&S_CMP=LP

- [Fer06] E.B.Fernandez and N. Delessy, "Using patterns to understand and compare web services security products and standards", Proceedings of the IEEE Int. Conference on Web Applications and Services (ICIW'06), Guadeloupe, February 2006.
- [Fer09] E. B. Fernandez, K. Hashizume, I. Buckley, M. M. Larrondo-Petrie, and M. VanHilst, "Web services security: Standards and products", to appear in "Web Services Security Development and Architecture: Theoretical and Practical Issues", Carlos A. Gutierrez, Eduardo Fernandez-Medina, and Mario Piattini (Eds.), IGI Global 2009.
- [For] Forum Systems, Sentry: Messaging, Identity, and Security, <http://www.forumsys.com/products/soagateway.php>
- [Gam94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994
- [Gnu] GnuPG, The GNU Privacy Guard, <http://www.gnupg.org/>
- [Has09a] K. Hashizume, E.B.Fernandez, and S. Huang, "Digital Signature with Hashing and XML Signature patterns", accepted for the 14th European Conf. on Pattern Languages of Programs, EuroPLoP 2009.
- [Has09b] K. Hashizume and E.B.Fernandez, "Symmetric Encryption and XML Encryption Patterns", sent to the Conference on Pattern Languages of Programs (PLoP 2009)

- [Has09c] K. Hashizume and E. B. Fernandez, “A Pattern for WS-Security”, submitted for publication.
- [IBM] IBM, WebSphere DataPower XML Security Gateway XS40, <http://www-01.ibm.com/software/integration/datapower/xs40/>
- [Inf04] W3C, XML Information Set (Second Edition), 4 February 2004, <http://www.w3.org/TR/xml-infoset/>
- [Ion] IONA Technologies, “Artix and Security.”
www.iona.com/info/aboutus/collateral/Artix%20and%20Security.pdf
- [Ker06] OASIS, Web Services Security: Kerberos Token Profile 1.1, 1 February 2006, <http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf>
- [Leh02] S. Lehtonen and J. Parssinen. “A Pattern Language for Key Management,” EuroPlop 2002. <http://www.hillside.net/patterns/EuroPLoP2002/papers.html>
- [Man08] DMTF, Web Services for Management (WS-Management) 1.0, 12 February 2008, http://www.dmtf.org/standards/published_documents/DSP0226_1.0.0.pdf
- [Met06] IBM, Web Services Metadata Exchange (WS-MetadataExchange) 1.1, August 2006, <http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf>
- [Mic07] Microsoft Corporation, .NET Framework Class Library, November 2007, <http://msdn.microsoft.com/en-us/library/ms229335.aspx>

- [Mica] Microsoft Corporation, .NET Framework Class Library,
<http://msdn.microsoft.com/en-us/library/e970bs09.aspx>
- [Micb] Microsoft Corporation, .NET Framework Class Library,
<http://msdn.microsoft.com/en-us/library/ms229749.aspx>
- [Mor06] P. Morrison and E.B.Fernandez, "The Credential pattern", Procs. of the
Conference on Pattern Languages of Programs, PLoP 2006, Portland, OR,
October 2006, <http://hillside.net/plop/2006/>
- [Mows06] OASIS, Web Services Distributed Management: Management Web Services
(WSDM-MOWS 1.1), 1 August 2006, [http://www.oasis-
open.org/committees/download.php/20574/wsdm-mows-1.1-spec-os-01.pdf](http://www.oasis-open.org/committees/download.php/20574/wsdm-mows-1.1-spec-os-01.pdf)
- [Mtom05] W3C, SOAP Message Transmission Optimization Mechanism, 25 January
2005, <http://www.w3.org/TR/soap12-mtom/>
- [Mul07] S. Mullan, Programming with the Java XML Digital Signature API, Sun
Microsystems March 2007,
http://java.sun.com/developer/technicalArticles/xml/dig_signature_api/
- [Muws06a] OASIS, Web Services Distributed Management: Management Using Web
Services (MUWS 1.1) Part 1, 1 August 2006, [http://www.oasis-
open.org/committees/download.php/20576/wsdm-muws1-1.1-spec-os-01.pdf](http://www.oasis-open.org/committees/download.php/20576/wsdm-muws1-1.1-spec-os-01.pdf)
- [Muws06b] OASIS, Web Services Distributed Management: Management Using Web
Services (MUWS 1.1) Part 2, 1 August 2006, [http://www.oasis-
open.org/committees/download.php/20575/wsdm-muws2-1.1-spec-os-01.pdf](http://www.oasis-open.org/committees/download.php/20575/wsdm-muws2-1.1-spec-os-01.pdf)

- [Nam06] W3C, Namespaces in XML 1.0 (Second Edition), 16 August 2006,
<http://www.w3.org/TR/REC-xml-names>
- [Not06a] OASIS, Web Services Base Notification 1.3 (WS-Base Notification), 1
October 2006, http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.pdf
- [Not06b] OASIS, Web Services Brokered Notification 1.3 (WS-Brokered
Notification), 1 October 2006, http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.pdf
- [OAS06] OASIS, Web Services Security: SOAP Message Security 1.1 (WS-Security
2004), 1 February 2006, <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>
- [OAS07] OASIS, W-S SecurityPolicy 1.2, 1 July 2007, <http://docs.oasis-open.org/ws-sx/ws-securitypolicy/v1.2/ws-securitypolicy.pdf>
- [Ope] The OpenSSL Project, OpenSSL, <http://www.openssl.org/>
- [PGP] http://en.wikipedia.org/wiki/Pretty_Good_Privacy
- [Pol07a] W3C, Web Services Policy 1.5 – Framework, 4 September 2007,
<http://www.w3.org/TR/ws-policy/>
- [Pol07b] W3C, Web Services Policy 1.5 – Attachment, 4 September 2007,
<http://www.w3.org/TR/2007/REC-ws-policy-attach-20070904/>
- [Rei06] B. Reistad et al, Web Services Resource Transfer (WS-RT) Version 1.0,
August 2006,

<http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-rt/ws-rt-spec.pdf>

- [Rel07] OASIS, WS-ReliableMessaging (WS-1 ReliableMessaging) Version 1.1, 14 June 2007, <http://docs.oasis-open.org/ws-rx/wsrn/200702/wsrn-1.1-spec-os-01.pdf>
- [Rel04] OASIS, Web Services Reliability (WS-Reliability) 1.1, 15 November 2004, http://docs.oasis-open.org/wsrn/ws-reliability/v1.1/wsrn-ws_reliability-1.1-spec-os.pdf
- [Res06a] OASIS, Web Services Resource Framework (WSRF) – Primer v1.2, 23 May 2006, <http://docs.oasis-open.org/wsrn/wsrn-primer-1.2-primer-cd-02.pdf>
- [Res06b] Web Services Resource 1.2 (WS-Resource), 1 April 2006, http://docs.oasis-open.org/wsrn/wsrn-ws_resource-1.2-spec-os.pdf
- [Res06c] OASIS, Web Services Resource Properties 1.2 (WS-ResourceProperties), 1 April 2006, http://docs.oasis-open.org/wsrn/wsrn-ws_resource_properties-1.2-spec-os.pdf
- [Res06d] OASIS, Web Services Resource Lifetime 1.2 (WS-ResourceLifetime), 1 April 2006, http://docs.oasis-open.org/wsrn/wsrn-ws_resource_lifetime-1.2-spec-os.pdf
- [Riv78] Ronald L. Rivest, Adi Shamir, Leonard M. Adleman: A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Commun. ACM* 21(2): 120-126 (1978).

- [RPo07] OASIS, Web Services Reliable Messaging Policy 2 Assertion (WS-RM Policy) Version 1, 14 June 2007, <http://docs.oasis-open.org/ws-rx/wsrmp/200702/wsrmp-1.1-spec-os-01.pdf>
- [RSA] RSA Security, PKCS #1: RSA Cryptography Standard, <http://www.rsa.com/rsalabs/node.asp?id=2125>
- [Saml05a] OASIS, Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, 15 March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>
- [Saml05b] OASIS, Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0, 15 March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-bindings-2.0-os.pdf>
- [Saml05c] OASIS, Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, 15 March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf>
- [Saml05d] OASIS, Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0, 15 March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-metadata-2.0-os.pdf>
- [Saml05e] OASIS, Authentication Context for the OASIS Security Assertion Markup Language (SAML) V2.0, 15 March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-authn-context-2.0-os.pdf>
- [Saml05f] OASIS, Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0, 15 March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-sec-consider-2.0-os.pdf>

- [Saml06] OASIS, Web Services Security: SAML Token Profile 1.1, 1 February 2006, <http://www.oasis-open.org/committees/download.php/16768/wss-v1.1-spec-os-SAMLSAMLTokenProfile.pdf>
- [Sch04a] W3C, XML Schema Part 0: Primer Second Edition, 28 October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>
- [Sch04b] W3C, XML Schema Part 1: Structure Second Edition, 28 October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/>
- [Sch04c] W3C, XML Schema Part 2: Datatypes Second Edition, 28 October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>
- [Sec04] OASIS, Web Services Security: SOAP Message Security 1.1(WS-Security 2004), 1 February 2006, <http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf>
- [SecP07] OASIS, WS-SecurityPolicy 1.2, 1 July 2007, <http://docs.oasis-open.org/ws-ss/ws-securitypolicy/v1.2/ws-securitypolicy.html>
- [Sig08] W3C, XML Signature Syntax and Processing (Second Edition), 10 June 2008, <http://www.w3.org/TR/xmlsig-core/>
- [Spml06] OASIS, Service Provisioning Markup Language (SPML) Version 2, 1 April 2006, <http://xml.coverpages.org/SPMLv2-OS.pdf>
- [SOA01] W3C, SOAP Security extensions: Digital Signature, W3C NOTE 06, February 2001, <http://www.w3.org/TR/SOAP-dsig/>
- [SOA07c] W3C, SOAP Version 1.2 Part 2: Adjuncts (Second Edition), 27 April 2007, <http://www.w3.org/TR/soap12-part2/>

- [Soap07a] W3C, SOAP Version 1.2 Part 0: Primer (Second Edition), 27 April 2007, <http://www.w3.org/TR/soap12-part0/>
- [Soap07b] W3C, SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), 27 April 2007, <http://www.w3.org/TR/soap12-part1/>
- [Sta06] W. Stallings, Cryptography and network security (4th Ed.), Pearson Prentice Hall, 2006.
- [Suna] Sun Microsystems Inc., Java Cryptography Extension (JCE), <http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html>
- [Sunb] Sun Microsystems Inc., Java SE Security, <http://java.sun.com/javase/technologies/security/>
- [Top06] OASIS, Web Services Topics 1.3 (WS-Topics), 1 October 2006, http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf
- [Tra06] W3C, Web Services Transfer (WS-Transfer), 27 September 2006, <http://www.w3.org/Submission/WS-Transfer/>
- [Trus07] OASIS, WS-Trust 1.3, 19 March 2007, <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html>
- [Uddi05] OASIS, UDDI Version 3.0.2, 3 February 2005, <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>
- [Use06] OASIS, Web Services Security: Username Token Profile 1.1, 1 February 2006, <http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-UsernameTokenProfile.pdf>

- [W3C01] W3C, XML Key Management Specification, March 2001
<http://www.w3.org/TR/xkms/>
- [W3C02] W3C, XML Encryption Syntax and Processing, 10 December 2002,
<http://www.w3.org/TR/xmlenc-core/>
- [W3C08] W3C, XML Signature Syntax and Processing (Second Edition), 10 June
2008, <http://www.w3.org/TR/xmlsig-core>
- [Wsci02] W3C, Web Services Choreography Interface (WSCI) 1.0, 8 August 2002,
<http://www.w3.org/TR/wsci/>
- [Wsd101] W3C, Web Services Description Language (WSDL) 1.1, 15 March 2001,
<http://www.w3.org/TR/wsdl>
- [Wsd101a] W3C, Web Services Description Language (WSDL) 1.1, 15 March 2001,
<http://www.w3.org/TR/wsdl>
- [Wsd101b] Web Services Description Language (WSDL) Version 2.0 Part 0: Primer, 26
June 2007, <http://www.w3.org/TR/wsdl20-primer/>
- [Wsd101c] W3C, Web Services Description Language (WSDL) Version 2.0 Part 1: Core
Language, 26 June 2007, [http://www.w3.org/TR/2007/REC-wsdl20-
20070626/](http://www.w3.org/TR/2007/REC-wsdl20-20070626/)
- [Wsd101d] W3C, Web Services Description Language (WSDL) Version 2.0 Part 2:
Adjuncts, 26 June 2007, <http://www.w3.org/TR/wsdl20-adjuncts/>
- [Wsd101e] W3C, Web Services Description Language (WSDL) Version 2.0 SOAP 1.1
Binding, 26 June 2007, <http://www.w3.org/TR/wsdl20-soap11-binding/>

- [Xacm05a] OASIS, eXtensible Access Control Markup Language (XACML) Version 2.0, 1 Feb 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
- [Xacm05b] OASIS, Core and hierarchical role based access control (RBAC) profile of XACML v2.0, 1 Feb 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf
- [Xacm05c] OASIS, Hierarchical Resource profile of XACML v2.0, 1 Feb 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-hier-profile-spec-os.pdf
- [Xacm05d] OASIS, Multiple Resource profile of XACML v2.0, 1 Feb 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-mult-profile-spec-os.pdf
- [Xacm05e] OASIS, Privacy policy profile of XACML v2.0, 1 Feb 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-privacy_profile-spec-os.pdf
- [Xacm05f] OASIS, SAML 2.0 profile of XACML v2.0, 1 Feb 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf
- [Xacm05g] OASIS, XML Digital Signature profile of XACML v2.0, 1 Feb 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-dsig-profile-spec-os.pdf
- [XIn06] W3C, XML Inclusions (XInclude) Version 1.0 (Second Edition), 15 November 2006, <http://www.w3.org/TR/xinclude/>

- [Xkms05a] XML Key Management Specification (XKMS 2.0) Version 2.0, 28 June 2005, <http://www.w3.org/TR/xkms2/>
- [Xkms05b] XML Key Management Specification (XKMS 2.0) Bindings Version 2.0, 28 June 2005, <http://www.w3.org/TR/2005/REC-xkms2-bindings-20050628/>
- [Xli01] XML Linking Language (XLink) Version 1.0, 27 June 2001, <http://www.w3.org/TR/xlink/>
- [Xml06] W3C, Extensible Markup Language (XML) 1.1 (Second Edition), 29 September 2006, <http://www.w3.org/TR/xml11/>
- [XQu07] W3C, XQuery 1.0: An XML Query Language, 23 January 2007, <http://www.w3.org/TR/xquery/>
- [Xtr] Xtradyne, Xtradyne's WS-DBC - the XML/SOAP Firewall for Enterprises, <http://www.xtradyne.de/products/ws-dbc/ws-dbc.htm>
- [XPa07] W3C, XML Path Language (XPath) 2.0, 23 January 2007, <http://www.w3.org/TR/xpath20/>
- [XPo03a] W3C, XPointer Framework, 25 March 2003, <http://www.w3.org/TR/xptr-framework/>
- [XPo03b] W3C, XPointer xmlns() Scheme, 25 March 2003, <http://www.w3.org/TR/xptr-xmlns/>
- [XPo03c] W3C, XPointer xpointer() Scheme, 19 December 2002, <http://www.w3.org/TR/xptr-xpointer/>
- [Xrml02] ContentGuard, XrML 2.0, March 2002, http://www.xrml.org/get_XrML.asp