摘 要

Web 服务提供了一种完全建立在现有互联网标准之上的面向服务的架构(SOA)，具有分布式、松散耦合性和平台无关性。随着 Web 服务的广泛应用，其安全问题已经越来越受到人们的重视。

安全性问题是一个非常复杂的问题，目前，与 Web 服务安全有关的规范主要有 WS-Security 规范、WS-Policy 规范、XKMS 规范以及 SAML 规范等，虽然这些规范在某些方面能实现消息安全，但不能提供一个完整的安全解决方案。Web 服务要求一种端对端的安全解决方案，包括加密机制、数字签名机制、安全管理、访问控制等等方面。

本文首先分析了 Web 服务的安全性需求，研究了 Web 服务安全，包括安全技术、安全性规范和最新发展。

然后，分析了 WSE3.0 的策略框架及工作过程，并对内置安全策略断言和自定义策略断言的应用分别进行了详细阐述。

其次，通过分析 WS-Security 安全模型及工作原理，针对 Web 服务的安全性需求，设计了一种基于策略架构的 Web 服务安全解决方案。本文给出了总体框架，并对安全消息交换的处理过程进行了描述，还指出了该方案的优点及不足。

最后，在.NET 环境下，使用 WSE 3.0 开发了一个 Web 服务安全应用系统。本文介绍了该系统的体系结构和针对安全需求的实现方法，描述了基本安全功能模块的实现，并详细阐述了高级安全功能模块的实现。

关键词：Web 服务安全；WS-security；WSE 3.0；策略

Abstract

Web Services provides a kind of services oriented architecture-SOA, Which is entirely built upon the current standards of Internet. It's distributed, loose coupling, and independent of platforms. With Web Services being used widely, its security has attracted more and more attention.

Security is a complex problem. At the present time, the specifications related to security of Web Services mainly have WS-Security, WS-policy, XKMS, SAML, etc. Though these specifications can implement security of message in some ways, they can not provide an integrated security solution. Web Services requires a security solution for end-to-end application, including encryption, digital Signature, security management, access control and so on.

First, this thesis analyzes security requirements of Web Services, and researches security of Web Services, including security technology, security specification and the latest development.

Then, this thesis analyzes the policy framework and the work process of Web Services Enhancements 3.0, and illuminates the application of turnkey security policy assertion and custom policy assertion respectively.

Next, through analyzing security model of WS-Security specification and its work principle, aiming at solving the security requirements of Web Services, a kind of Web Services security solution based on policy architecture is designed. This thesis provides the whole framework, describes the process of security message exchange, and indicates the advantages and the disadvantages of the solution.

Finally, a Web Services security application system is developed on .NET platform with WSE3.0. This thesis introduces the architecture of the system and the realization methods aiming at security requirements, describes the realization of the basic security function modules, and illustrates the realization of the advanced security function modules.

Key words: Web Services Security;WS-security;Web Services Enhancements 3.0; Policy

第1章 绪论

1.1 Web服务安全的研究背景

Web 服务技术基于许多不同软件应用程序的互操作性，而这些应用程序通过 Internet 在各地各种系统中运行，通过使用 XML、SOAP、UDDI、WSDL 以及其他协议和机制，实现跨域且独立于平台的交互作用。正是由于Web服务这种分布式、异构的本质[1]，使得Web服务的安全变得很复杂。而Web服务开发初始希望其简单易用，最初设计者选择了推迟定义解决安全问题的方法[26]。

目前安全技术已相当普及，各部门都会建立自己的特定安全解决方案，但这些解决方案往往具有局部性。而在设计处理Web服务模型的新解决方案时，由于系统实现者希望能充分利用现有的投资，并不取代现有的安全解决方案，而是想将这些独立的解决方案整合为一个完整的端对端安全解决方案。

因此，Microsoft 和其他业界企业（例如 IBM、BEA）一起制定了一组为实现丰富和可互操作的通讯协议和规范，通常称作 WS-* 规范。该规范及其后续的工作就成为了Web服务技术领域的一次最重要的进步。各软件服务商为其产品提供相应的接口和编程工具。Web 服务开发者也将利用这些工具来加强所开发的Web服务的安全性。

微软使用 WS-Security 等规范实现 SOAP 消息安全的工具包最初为 Web Services Enhancements 1.0 for .net （简称为 WSE 1.0），它作为一个类库用来实现高级Web服务协议，在保护Web服务安全中一个重要的内容就是要保护其SOAP消息传递的安全。通过使用WSE，使SOAP消息可以自己验证其完整性，并可利用规范中的机制进行签名和加密。

在2005年底微软推出了Visual Studio 2005版本，其中WSE推出了3.0版本[25]，并直接构建到 Visual Studio 2005 中，作为Web服务的安全性开发工具，WSE（1.0和2.0）推出的主要目的是提供一种实际可行的设计方法[18]，来实现WS-*安全性规范，而3.0版本的目的不是去改变规范，而是因为Web服务已应用到许多开发领域，它必须扩充 Visual Studio 中现有的 Web 服务支持，解决、简化开发人员所面临的问题。其主要提供的目标有：简便

开发安全的Web服务、采用Web服务协议与.net 2.0结合简化面向服务的应用开发以及实现与未来技术的兼容性。

目前，关于 Web 服务安全技术的理论研究已经较多，而且也出现了用WSE1.0或2.0设计实现Web服务安全体系，虽然各有侧重点，但也为下一步研究提供了理论和实践的基础。而微软刚推出的 WSE 3.0 开发工具包[7]，为研究提供了很好的开发平台以及开发工具。Web 服务应用在现阶段得到了大力发展，这方面的新技术和新产品也在不断涌现，不断为研究提供新的素材。

1.2 本课题研究工作的意义

1. WSE 3.0是最新推出的Web服务安全开发工具包，以前的研究还只是利用1.0或2.0进行开发，并未对3.0的技术展开探讨。

2. 以往的研究主要是侧重Web服务安全技术的理论探讨，以及研究如何基本实现Web服务的安全。而目前的研究应该再进一步，研究如何更加便利的开发Web服务安全体系，如何提高Web服务安全实现的效能以及如何与未来安全技术规范接轨问题等方面。

1.3 Web服务安全的研究现状

标准是维护Web服务应用程序可移植性和互操作性的最好方法。虽然产品和技术在不断变化，但其基础安全服务所普遍采用的安全标准将很稳定。因此，对于采用了标准的安全应用程序编程接口的应用程序，就可在不重写应用程序的情况下，开发自己的安全产品。

目前，WS-Security 说明书已经准备被批准，并有很多安全产品都在支持它，然而，现在对配置安全还没有标准。WS-Policy 和 WS-Trust 这两种 WS* 说明书也已经被提交到标准体系中，它们有助于解决安全配置问题。

现在已有厂商支持WSS规范和Web服务安全功能。支持WSS的产品必须包含一个WSS提供器（能处理WSS SOAP头）。一般可归入如下几类:Web服务平台、WSS库、Web服务管理(Web services management，WSM)、XML安全网关、XML虚拟专用网、Web服务欺骗探察以及Web服务单点登录及联邦[26]。而XML安全网关和Web服务管理产品提供了使安全性外部化，以使它能被集中

地管理和控制的方法，是目前最好的解决办法，也是过渡期研究的主要方向。

1.4 本论文研究的主要内容

在本文中主要的研究目标是：针对目前Web服务的安全需求，依照WS-*规范，利用.NET 2.0的开发平台，采用WSE 3.0的开发工具，设计一种基于策略架构的Web服务安全解决方案。并开发一个Web服务安全应用系统，实现该方案。

因此，研究中的主要内容是：

1. 对WSE 3.0中的策略体系架构及工作过程进行研究，尤其是自定义策略断言的应用。

2. 利用WSE 3.0开发工具，针对Web服务的安全性需求，设计一种Web服务安全解决方案。主要考虑如何采用策略文件与代码相结合的方式来设计SOAP消息处理模块。

在设计解决方案时，一方面从端对端企业角度（从上至下）考虑，先对框架的整体进行设计，再细化各实现模块的功能；另一方面从身份认证、授权、安全消息交换这些主要的安全技术的实现角度（从下至上）考虑，先设计各模块的实现功能，再考虑如何组合到策略中去。

3. 建立并逐步实现适合自身需要的应用平台的Web服务安全体系框架，实现各种安全功能，满足各种不同的安全需求。

拥有一个安全架构，不是要一次实现所有安全功能。这里也是从小规模开始，先选择基本的安全服务，然后再在需要的时候建立更加高级的安全功能。而架构为建立整体安全体系提供了实现路线图。

1.5 本章小结

在应用程序内自定义建立的安全是手工编码，实现和维护起来费用较高，而且还可能存在更多的安全脆弱性，而一个单一的、采用了标准的应用程序编程接口的安全基本结构则可以被所有应用程序使用，可避免多次重复定义用户、安全属性和其他策略，可以集中精力解决一些关键的可互操作的安全技术。因此，建立一个跨企业共享的通用安全结构是值得进行探讨的。

第2章 Web服务安全技术

本章讨论Web服务安全的相关技术，为设计和实现本论文的Web服务安全架构做好理论铺垫。

2.1 Web服务安全及目标

随着Web服务在松散耦合的、与语言和平台无关的应用中的采用越来越广泛，另一个潜在问题也日益引起人们的关注—安全性。Web服务的安全性对商业组织和它们的客户来说都是至关重要。在向Internet开放重要业务功能的过程中，Web服务也把企业内部数据、应用程序和系统暴露给了各种外部威胁，如信息窃取、欺诈、破坏等行为。这给企业信息安全带来了极大的隐患。如果信息安全问题不能被妥善解决，Web服务的推广应用将会大大延缓。而Web服务的安全领域却是有待开发的领域。

安全目标包括：

（1）端到端的消息传递：即消息级别的安全性，不仅在传输的过程中要保证消息是安全的，而且在到达终点之前的各个中介处也需要是安全的。

（2）提供机密性和完整性的安全消息交换：机密性：加密消息，防止消息在传输过程中被泄露。完整性：确保消息在传输的过程中不被篡改。

（3）身份认证和授权：身份认证：通过验证用户所拥有的可靠凭证来确保其身份与消息中所声称的用户身份一致，避免冒名者。凭证可以是口令、X.509数字证书或任何一种安全令牌。授权：用户身份被验证通过后，就可根据访问权限授予对资源的访问权。

（4）不可抵赖性：这在电子商务中是不可豁缺的[21]。确保恶意发送方在事后无法抵赖其创建并发送特定消息的事实，避免遭受恶意第三方如重放攻击等技术的袭击。

（5）消息安全中的灵活性：由于不同的用户对Web服务有不同的访问权限，需要提供对SOAP消息的细粒度保护，支持如部分加/解密等特殊安全处理。

这些问题虽然与国际标准化组织ISO在网络安全体系的设计标准中定义的安全服务功能类似[22]，但在实质上还是有区别。

2.2 Web 服务所用的安全技术

Web 服务是工作在应用层，而以前的安全技术和产品基本上是工作在网络层或传输层。Web 服务经常跨越不同系统和平台，而以前安全问题的解决基本是在一个平台甚至一个系统上。现在已经提出了一些技术，以使 Web 服务更加安全。

2.2.1 传统的 Web 安全技术

1. 防火墙技术和 IP 安全协议（IPSec）

属于网络级别的安全技术，根据策略和规则的设置，限制已知 IP 访问，保护专用网络不受外部网络的入侵，但是 Web 服务结合使用 HTTP 协议与 XML，实现的是应用程序之间的相互通信，防火墙只能放行或滤掉所有的 SOAP 消息，其内容一般不会被检查，因此 Web 服务能轻易穿越防火墙。防火墙技术并不适用于应用级别的安全性。

2. 安全套接字层（Security Socket Layer, SSL）

属于传输级别的安全技术，为目前业界现成的而且广泛接受的传输层安全机制。但只有在点对点的情况下才能在传输过程中提供消息完整性和机密性； 但 Web 服务所发送的 SOAP 消息为端对端的消息传递，在到达最终目的地或接收方之前一般要经过一个或多个中介体（如应用网关）接收并处理，而 SSL 无法提供在中介体传输层以上的某一层之间的安全。因此，无法满足 Web 服务开发和使用的安全要求。

下面通过一个基本示例进行比较说明，当 Web 服务请求者通过 Web 客户端与 Web 服务器（直接提供 Web 服务）进行消息传递时，如图 2-1 所示。这时为点对点的消息传递，Web 客户端与 Web 服务器之间可通过 SSL 建立连接。Web 服务安全性可由 SSL 提供。

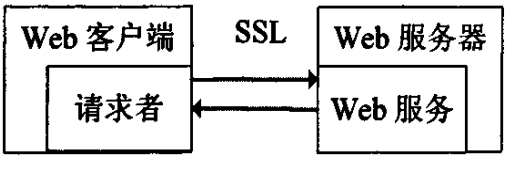


图 2-1 点对点的消息传递

但当Web服务器本身不直接提供Web服务，而只作为一个中间层Web应用程序，Web服务请求者所请求的Web服务是由其他应用程序提供时，由于消息需要经Web服务器（中间层Web应用程序）处理后，才能再向目标Web服务传递，如图2-2所示，这时为端对端的消息传递，而SSL仅能提供传输层安全，因此，在中间层的应用层安全就无法提供。导致请求者与Web服务提供者之间的秘密就可能在中间层Web应用程序中被泄漏。

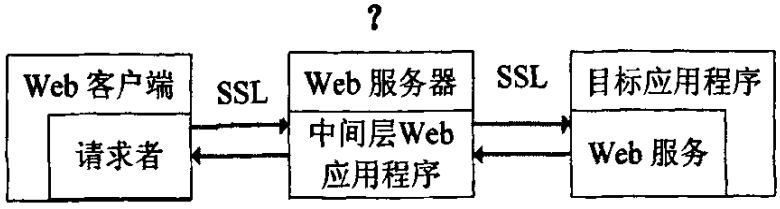


图2-2 端对端的消息传递

通常，将传统安全技术作为Web服务的第一层保护，同时采用更高层的安全技术来处理Web 服务的安全需求。

2.2.2 目前所用的安全技术

Web 服务所用的安全技术属于应用级别的安全技术，提供端对端的消息级安全。

所涉及的安全技术有：

1. 基本密码术

密码术是所有安全体系结构的基础，主要分两类：对称密码术和非对称密码术。对称密码术常用来加密批量消息数据[2]。由于非对称密码术在加解密时需要进行复杂的计算，速度比对称密码术慢很多。所以常只用来加密很少数量的信息，如一个DES对称密钥。然后再用该对称密钥来加密消息。

为了能运行非对称算法，需要采用某种方法发布公钥。一般通过证书颁发机构提供CA服务，建立公钥基本结构PKI，为用户提供数字证书用来证明其身份。使用户可以确信所使用的公钥是与它的所有者相关联的。证书可以从可信的第三方获得，也可以通过在公司内部建立本地可信的CA服务器来提供。

2. 数字签名

就是先应用密码散列算法（常用的有SHA算法和MD5算法）对消息生成

散列值[6]，再用数字签名算法（常用的有RSA算法和DSA算法）对散列值实现数字签名。能有效地检测消息是否被篡改。

3. XML加密技术

由W3C规范清晰地、完整地定义了XML加密语法[19]，并规定了加密整个或部分XML文档的方法。

XML加密为加密数据和以标准XML格式表示加密结果提供了一种标准的方法。XML加密允许加密任何数据，可以是一个完整的XML文档或一个XML文档中的指定元素，也可以是从外部引用的任意非XML格式数据，或间接地从外部引用加密的数据，并支持多重加密。

4. XML签名技术

由W3C定义，以XML模式的形式提供了XML签名的语法和语义[58]。用于对以XML格式表示的数据进行数字签名。描述了数字签名的XML表示及计算、验证数字签名的过程。

可以将XML签名和XML加密两种技术结合在一起对一个XML文档进行操作[16]。这时两种操作的顺序就要注意。应用程序必须区别加密操作在签名操作前还是在签名操作后完成。

2.3 Web服务安全性规范

为解决Web服务在处理消息级别的安全问题，Microsoft和其他业界企业（例如IBM、BEA）一起制定了一组为实现丰富的和可互操作的通讯协议和规范，通常称作WS-*规范[1]，主要是在Web服务使用SOAP（XML格式）作为消息传输协议的背景下，将XML签名、XML加密和SAML(Security Assertion Markup Language: 安全声明标记语言),组合起来以满足Web服务安全需求的一套规范。该规范与传输层无关，可以使用HTTP, TCP等协议，并通过只使用和应用相关的协议和规范来实现可组合性。

这组规范主要包括消息安全性模型（WS-Security）、策略模型（WS-Policy）、信任模型（WS-Trust）和隐私权模型（WS-Privacy）等。这些初始规范结合在一起提供了一个基础，在这个基础上可以跨多个信任域来建立安全的、可互操作的Web服务。如图2-3所示：

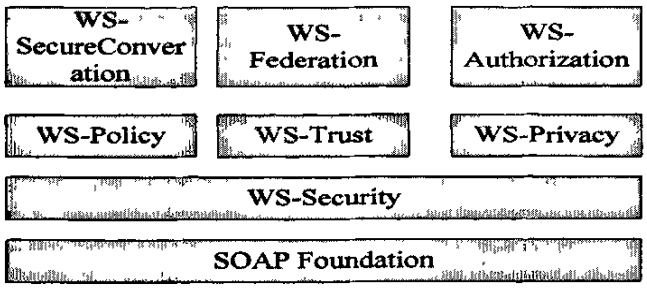


图 2-3 WS-*规范层次结构

1. 消息安全性模型（WS-Security）

WS-Security 规范实际上是对 SOAP 协议的扩展。规范利用了多种常见的、现有的安全性标准和规范，本身并没有提出新的加密算法或安全模型。通过提供一个架构，将这些现有的技术注入到 SOAP 消息中。使用户可自由地将 Web 服务协议、应用层协议与各种加密技术、安全模型结合起来，以实现 Web 服务环境下消息的完整性、保密性和消息验证。

SOAP 规范本身没有指定安全性功能，而是由 SOAP 头提供可扩展机制，以扩展 SOAP 消息使其可以适用于多种用途。因此可以通过对 SOAP 头元素的可扩展性处理来实现安全性功能。在 WS-Security 规范中据此定义了向 SOAP 添加安全性的基本方法。通过定义一个用于携带安全性相关数据的 SOAP 标头<Security>元素来实现。

WS-Security 还提供了三种主要机制：

（1）安全性令牌

包含由现有安全机制定义的安全性信息，形成令牌是为了传递的一致性规范，使之能在所有层之间传递并统一识别。当对 SOAP 消息进行加密或签名时，必须传递安全性令牌给接收者，以便于接收者对消息进行解密或验证签名。同时安全性令牌可以传送关于发送者的身份信息。

在这个标准机制中，安全性令牌被包含在 SOAP 消息头<Security>元素中，并以明文形式存在，需要使用消息完整性和消息机密性来保证其安全性。WS-Security 非常灵活，并支持多种安全性令牌。如：用户名/口令令牌、X.509 证书令牌和 SAML 令牌。

（2）消息完整性

使用 XML 签名规范来实现该功能。支持多种签名算法、多种签名格式。

（3）消息机密性

通过同时使用 XML 加密和把安全性令牌作为 SOAP 消息的一部分来实现。

加密机制支持多种加密算法、多种加密格式。加密也可以引用一个安全性令牌。

上述三种机制可以单独使用(例如，传输安全性令牌)，也可以组合使用，以实现不同强度的安全性。(例如，对消息进行签名和加密，并提供与签名/加密密钥相关的安全性令牌)。

2. 策略模型（WS-Policy）

描述了中介体和端点必须满足的安全策略。

3. 信任模型（WS-Trust）

描述了使Web服务能够安全地进行互操作的信任模型。

4. 隐私权模型（WS-Privacy）

描述了Web服务中如何定义个人隐私权策略。

5. 安全对话模型（WS-SecureConveration）

描述如何管理和验证各方之间相互交换的消息。

6. 联合模型（WS-Federation）

描述如何管理和协调异类环境中的信任关系。

7. 授权模型（WS-Authorization)。

定义了Web服务管理验证数据和授权策略的方法，在授权格式和授权语言上是灵活的且可扩展的。

安全模型是在扩展的SOAP协议支持的传输层的基础上引入Web服务安全层。Web服务安全层中的WS-Security协议已经定稿，而上面的两层协议至今为止还未确定下来。WS-Security协议是构成整个Web服务安全层的基础。满足了WS-Security协议也就构成了最初的Web服务安全层。WS-Security协议与其上层的WS-Policy 、WS-Trust和 WS-Privacy协议构成了可信环境中的Web服务安全层。上述四个协议加上其上层的三个协议构成了不可信环境中的Web服务安全层，也就是整个Web服务安全层。

2.4 本章小结

本章讨论了在 Web 服务安全架构的设计中使用到的几个关键技术及WS-*规范，为下一步的设计打下了理论基础。

第3章 WSE3.0策略框架技术分析

本章对WSE3.0策略框架技术进行分析，该技术是目前比较前沿的研究技术，其中的概念、方法及处理流程为本论文中解决方案的设计和实现提供了重要的参考价值。

3.1 WSE的发展

如果没有良好的安全性，Web服务就不能发挥它的作用。Microsoft和IBM以及其他公司共同制定了一组提供Web服务安全性的规范，其中最重要的是WS-Security。现在，WS-Security规范系列在很大程度上已日臻完善[25]。Microsoft发布Web服务安全性开发工具包的3.0版（Web Services Enhancements又名WSE），主要目标就是为开发人员提供WS-*规范的第一个完整实现,用来开发符合WS-Security规范的消息级安全性解决方案。

作为一个用最新Web services协议来建立Web services的.NET类库。其目的不是去改变规范，而是因为Web服务已应用到许多开发领域，它必须扩充Visual Studio中现有的Web服务支持，解决、简化开发人员所面临的问题。

3.2 WSE所支持的安全特性

WSE要保护Web服务安全,其中一个重要的内容就是要使用WS-Security等规范实现SOAP消息安全。主要分为以下三个方面：

1. 安全凭证

提供端对端的身份认证。所使用的机制在WS-Security规范中定义，就是在SOAP消息内放置安全凭证。客户端从可信第三方获得安全凭证。通过支持WS-Trust，WS-SecureConversation，和基于XML令牌的WS-Security Profile规范，WSE 3.0扩展了给SOAP消息添加安全凭证的概念。

2. 完整性

通过对SOAP消息进行数字签名以及接收方验证该签名，让SOAP消息的

接收方可以检查 SOAP 消息在传递过程中是否被修改。WS-Security 标准要求数字签名遵循 XML 数字签名标准。该标准支持对 SOAP 消息的选定部分进行签名。

3. 机密性

通过加密 SOAP 消息，使消息发送方可以确保 SOAP 消息内容仅对预定的接收方可见。WS-Security 要求对 SOAP 消息的加密遵循 XML 加密标准，该标准支持对 XML 文档的选定元素进行加密。

3.3 WSE3.0 的策略框架

在 WSE3.0 中，实现所支持的安全特性是基于可扩展的 WSE 3.0 策略框架。WSE 策略框架提供了一种机制，以描述 Web 服务需要执行的约束和要求[30]。

WSE 3.0 使用策略来创建整个管道。如图 3-1 所示，一个策略包含了一个有序的策略断言列表[55]。每个策略断言定义一个对 Web 服务的要求[23]。并在策略编译过程中，由每个策略断言生成的筛选器负责对进入和离开终结点的 SOAP 消息进行截获和处理，来执行对 Web 服务的要求。策略断言生成筛选器遵循策略中断言的顺序。而管道表示这个输入筛选器和输出筛选器的集合。

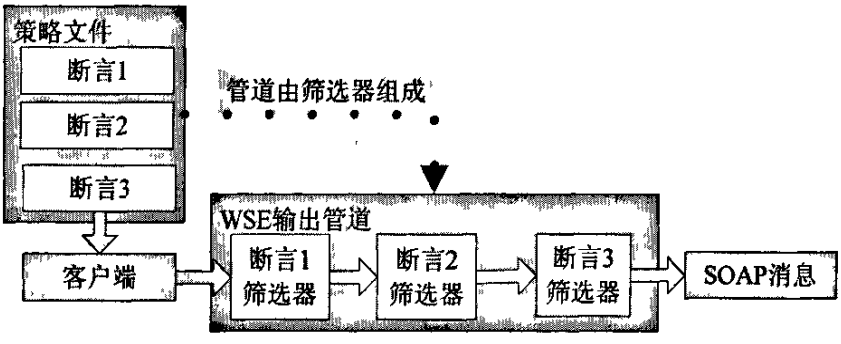


图 3-1 WSE3.0 输出管道中的策略

该框架通过使策略断言和管道筛选器成为一体，提供了一个用于配置和扩展 WSE 行为的统一模型。

3.3.1 相关概念

1. 策略

开发者通过在一个XML格式的配置文件内对请求和响应SOAP消息公开声明需求，这些需求被共同看作为一个策略，所有策略组成一组被命名的策略集，来对应包括SOAP消息交换所需的安全需求，以及其他需求，如跟踪。

该配置文件被称为策略文件。在应用中，可由Web服务提供方统一创建对应Web服务端和客户端的策略文件，并分别提供给两端使用，从而建立统一的安全通道。

2. 策略断言

每个被命名策略包含了一个有序的策略断言集合。策略断言描述了为在客户端和Web服务端之间的SOAP消息交换所定义的一组需求。例如，一个策略断言可指定一种加密SOAP消息的安全令牌类型。在初始化服务或服务代理时，WSE 3.0会检查提供的策略，并实例化各策略断言。

3. 筛选器

由策略断言遵循在策略中的顺序生成，负责对进入和离开端点的 SOAP 消息进行截获和处理，来执行对 Web 服务的需求。

在策略编译过程中，每个策略断言最多可以生成四个筛选器。每个筛选器在不同的请求/响应消息交换阶段执行。两个不同的筛选器处理客户端上的请求消息和响应消息，两个不同的筛选器处理服务端上的请求消息和响应消息。该简单体系结构如图3-2所示：

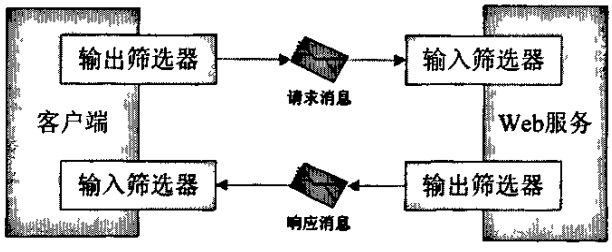


图3-2 WSE筛选器简单体系结构

通常，筛选器只对发送和接收的SOAP消息的头部进行处理。根据功能的不同，筛选器分为两种：输出筛选器和输入筛选器。输出筛选器负责根据需求对客户端发送的请求消息或服务端发送的响应消息，写入相应的SOAP消息头；而输入筛选器负责对服务端接收的请求消息或客户端接收的响应消息读

取 SOAP 消息头并检查其有效性。此外，筛选器还可以根据需要对消息体进行处理。例如，加密发送的 SOAP 消息体和解密接收的 SOAP 消息体消息。

4. 管道

筛选器的集合则表示为管道。如果策略没有被提供，则创建一个默认管道。策略和策略断言作为部署时概念分别对应管道和筛选器运行时概念。

由于策略由一个或多个有序策略断言组成，并且每个断言都会在管道中插入对在客户端和Web服务端之间交换的SOAP消息执行处理的一套输出和输入筛选器。因此，通过策略内策略断言的顺序，可以控制管道中筛选器在运行时的构建顺序，从而控制运行时应用或执行有关要求的顺序。

在消息实际到达 Web 服务方法之前，在管道中执行安全功能，使应用程序开发人员能方便地实现安全。这有助于在所有 Web 服务方法中应用一致的安全策略。

3.3.2 策略对象模型

通过阐述策略框架背后的对象模型有助于加深对上述概念的了解。

1. 策略断言相关类

主要有以下一些：

PolicyAssertion ：是策略断言相关类的抽象基类。

MutualCertificate11Assertion：为内置安全断言类的一种。

SecurityPolicyAssertion：为自定义安全断言类。

每个策略断言类中，除了构造函数和 4 个用于创建 SOAPFilter 对象的方法外，主要包括 ReadXml 方法和 GetExtensions 方法。

ReadXml 方法：主要用于读取策略文件中与该策略断言相关的 XML 元素。对于内置安全断言或从中继承的自定义断言，由于本身会自行进行读取，所以不需要重载该方法，对于没有子元素的自定义断言因为没有内容，也不需要重载，而对于不能自动读取，又有子元素的自定义断言就需要重载该方法来分析处理自定义的断言元素及其中内容。

GetExtensions 方法：实现获取在策略文件中被注册的策略扩展集。

2. 筛选器相关类

筛选器的行为由策略断言定义，并且是有序的。每个断言对应两组筛选器，四个筛选过程。主要有以下一些：

SoapFilter：是一个一般类，为运行时类。负责处理客户端或 Web 服务端上的请求或响应 SOAP 消息。

ReceivesecurityFilter：输入安全筛选器类，专门负责接收时对 SOAP 消息的安全处理。

SendSecurityFilter：输出安全筛选器类，专门负责发送时对 SOAP 消息的安全处理。

SoapFilterResult：保存一个筛选器处理后的 SOAP 消息的数据，并提供给下一个筛选器处理。

3. Policy 类

包含 PolicyAssertion 类，充当创建客户端管道或服务管道的工厂。

4. Pipeline 类

代表已排序的 SOAPFilter 对象集合。

3.3.3 WSE 筛选器的处理过程

WSE 筛选器处理链已经集成到带 WSE 的 SOAP 通信中，并且是 ASP.NET Web 服务的基本结构。其处理过程描述如下：

1. 与 Web Service 代理的集成(客户端一方)

WSE 输入和输出筛选器通过一个名为 WebServicesClientProtocol 的代理基类提供给 Web services 客户端[7]。

(1) 输出筛选器的处理过程

通过 WebRequest 类完成，WebRequest 实例对每一个准备发送的 SOAP 请求信息进行分析，生成 SoapEnvelope 类的一个实例，WebRequest 类有一个属性类 SoapContext， 负责存放对 SOAP 消息的特定处理需求(如：加密、签名、数字证书与时间戳等)。通过策略中的策略断言的控制，输出筛选器的处理链依次传递 SoapEnvelope，并由各个输出筛选器按顺序根据 SoapContext 类实例对请求信息进行处理，形成特定的 SOAP 消息头，最终形成符合需求的 SOAP 消息并发送。如图 3-3 所示：

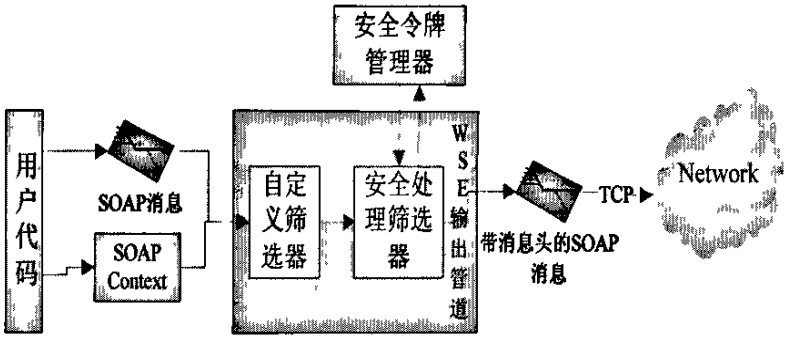


图 3-3 输出筛选器的处理过程

（2）输入筛选器的处理过程

通过 WebResponse 类完成，WebResponse 类对 SOAP 消息的处理与 WebRequest 类相反。它将客户端收到的 SOAP 响应消息进行解析，形成 SoapEnvelope 类的一个实例，然后通过输入筛选器处理链依次传递。由每个输入筛选器有序进行消息头的有效性验证和消息体的处理(例如解密)。与 WebRequest 一样，这套筛选器也由与 SOAP 请求相关联的策略定义。WebResponse 类也有一个属性类 SoapContext，在处理特定的 SOAP 消息头时，相应地会将特定设置(加密、签名、数字证书与时间戳等)存放在 SoapContext 里。最终一个基本的 SOAP 消息将传送给客户端。如图 3-4 所示：

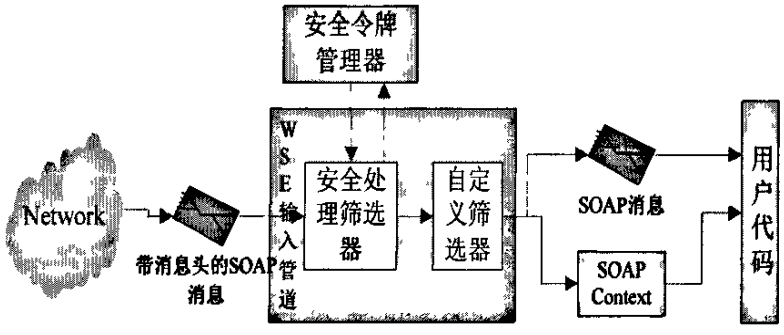


图 3-4 输入筛选器的处理过程

2. 与 Web Services 的集成（服务端一边）

在服务端也有一套和客户端功能相似的输入/输出筛选器，通过服务器一边的 SOAP 协议工厂类 WseProtocolFactory 提供给 ASP.NET Web 服务。WseProtocolFactory 类负责截获发送给 Web 应用程序中 Web 服务的 SOAP 消息，处理符合 WS-*规范的 SOAP 消息头。确保 Web 服务的方法被调用时，WSE 筛选器能处理与服务端交互的 SOAP 消息。

3.3.4 安全会话

当客户端需要与 Web 服务端传递大量的 SOAP 消息时，则在两者之间建立安全会话是非常有用的。首先由客户端向 Web 服务端请求安全会话令牌(Secure Conversation Token，SCT)，而由 Web 服务端向客户端提供 SCT，只要该 SCT 未到期或未被取消，以后就可用这个 SCT 签名和加密在两者之间传递的一系列消息。这被称为一个在客户端和 Web 服务端之间的安全会话。

建立安全会话的过程如图 3-5 所示，

1. 客户端向被 Web 服务端的安全令牌服务发送它的第一条 SOAP 消息，被称为请求安全令牌（Request Security Token，RST）消息，来从安全令牌服务中请求一个安全会话令牌。

2. 默认情况下，这个 RST 必须被签名，以至于安全令牌服务能验证客户端是有资格来接收这个安全会话令牌的。在签名被验证、确认和授权后，安全令牌服务返回一个包含会话令牌的 SOAP 消息，该消息被称为请求安全令牌响应（Request Security Token Response，RSTR）消息。

3. 用这个安全会话令牌在客户端和目标 Web 服务之间提供安全性。

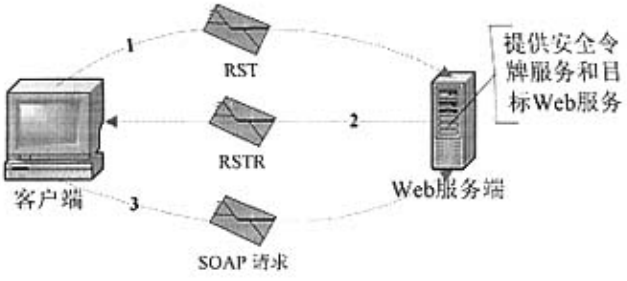


图 3-5 安全会话建立过程

安全会话令牌是基于对称密钥，这比非对称密钥在数字签名或加密 SOAP 消息方面更有效率。

WSE3.0 提供的安全令牌服务和安全会话令牌支持 WS-Trust 和 WS-SecureConversation 规范。

3.4 策略断言的分类

3.4.1 内置安全策略断言

以往应用程序的安全经常是通过多重低层次安全操作来共同拼凑。针对此问题，WSE3.0 允许使用预先包装好的一套安全操作在较高层次应用安全。对应最常见的情况和方案，WSE 3.0 中的策略框架提供了六个内置安全策略断言。这是WSE3.0中最重要的新增安全功能[32]。

选择内置安全策略断言来实现安全策略，可以使一部分安全问题在部署时期进行考虑。一种内置安全策略断言代表了一个场景，这六个场景都是基于行业中的最佳实践而总结出来的。例如：应用场景之一：客户端和WEB服务应用程序在Internet网中通信，如图3-6所示：

用Web服务端的
X.509证书实现消息机密性

图3-6 在Internet 之上通信的应用程序

这时,所适用的安全断言之一为：<usernameForCertificateSecurity>。

3.4.2 自定义策略断言

内置安全断言只支持最常用的安全需求情况，并不满足所有的需求，这就需要创建自定义策略断言来强迫实施安全需求或其它需求。WSE3.0提供了创建与 WSE 提供的现有声明性安全策略相结合的自定义声明性策略的扩展功能。这是一种用于向 Web 服务中添加自定义需求的方式。

要实现自定义安全断言，不仅要创建自定义安全断言类，还要创建自定义安全断言中使用的自定义筛选器类，并实现在管道中插入自定义筛选器，在策略中使用断言类，然后将该策略应用于服务。

创建的基本过程为：

1. 创建一个从 SecurityPolicyAssertion 类派生的自定义安全断言类。

- 重载 CreateServiceInputFilter 方法来指定 Web 服务的自定义输入筛选器；
- 重载 CreateServiceOutputFilter 方法来指定 Web 服务的自定义输出筛选器；
- 重载 CreateClientInputFilter 方法来指定客户端的自定义输入筛选器；
- 重载 CreateClientOutputFilter 方法来指定客户端的自定义输出筛选器。
- 重载 ReadXml 和 GetExtensions 方法解析策略文件中的自定义 XML 元素和属性。

2. 为 Web 服务和客户端创建用于自定义断言类的自定义筛选器类。

- 为客户端创建一个从 SendSecurityFilter 类派生的自定义输出筛选器类。重载 SecureMessage 方法保护从客户端发送的 SOAP 请求。在自定义筛选器的构造函数中创建安全令牌，使用该安全令牌处理 SOAP 请求，并将安全令牌保存在 OperationState 属性中，以后提供给客户端的自定义输入筛选器访问，来验证从 Web 服务返回的 SOAP 响应中的安全需求。
- 为 Web 服务创建一个从 ReceiveSecurityFilter 类派生的自定义输入筛选器类。重载 ValidateMessageSecurity 方法对 Web 服务接收的 SOAP 消息验证消息所指定的安全需求。通过处理接收的 SOAP 消息，获得安全令牌，并将安全令牌保存在 OperationState 属性中，如果 SOAP 响应返回给客户端，则以后可提供给 Web 服务的自定义输出筛选器访问。
- 为 Web 服务创建一个从 SendSecurityFilter 类派生的自定义输出筛选器类。重载 SecureMessage 方法保护从 Web 服务发送的 SOAP 响应。安全令牌从 Web 服务的自定义输入筛选器已设置的 OperationState 属性中获得。
- 为客户端创建一个从 ReceiveSecurityFilter 类派生的自定义输入筛选器类。重载 ValidateMessageSecurity 方法来对 SOAP 响应验证其安全需求。安全令牌从客户端的自定义输出筛选器已设置的 OperationState 属性中获得。

自定义筛选器类必须从SoapFilter基类中派生。并重载SoapFilter基类中的ProcessMessage 抽象方法。

如果自定义筛选器涉及安全性，则需要从 ReceiveSecurityFilter 或SendSecurityFilter 派生，这取决于通信的方向。这两个类派生自SoapFilter，它们提供与消息安全相关的额外方法。ReceiveSecurityFilter定义一个抽象的 ValidateMessageSecurity 方法，SendSecurityFilter 定义一个抽象的 SecureMessage 方法。这两个方法提供 Security 对象，可以使用该对象执行命令性安全任务。

若自定义策略断言不涉及安全性，则可建立一个从PolicyAssertion类派生的自定义策略断言类 。对应的自定义筛选器类可直接从SoapFilter类派生。并通过重载ProcessMessage方法来变换或查看SOAP消息。如实现跟踪。

自定义策略断言体系结构提供了生成安全基础结构所需的灵活性，从而使开发人员随后可以在他们的应用程序中轻松地通过策略来重用这些安全基础结构。

3.5 应用策略

在WSE中，策略的应用允许在两种模型中实现。一种是基于代码的模型。策略在代码中以命令方式定义，由开发人员指定策略要求。另一种是指定安全性配置的策略模型。策略在外部 XML 文件中以声明方式定义。开发人员在代码中只提供应用程序逻辑，而将指定策略要求的责任委托给管理员。

在WSE的先前版本中，用于指定安全性配置的策略模型与基于代码的模型是分开的。而在 WSE 3.0 中，这两种技术的实现模型是相统一的，通过用代码创建策略断言时，所提供的参数与通过策略文件提供的参数保持了一致，实现了配置与编码的完整结合。

在大多数 Web 服务部署中，当策略要求依赖于部署环境且不能在开发时确定时，使用策略文件来描述安全性要求使管理员有了所需的灵活性，即使服务的安全环境发生变化，也可以确保 Web 服务的可访问性和安全性。而使用代码一般是要在开发环境已知并且不再改变的情况下，并且要通过指定策略来实现。尤其在实现某些新的内置安全断言时，重新部署总会带来一定的

复杂性。而这时一般不愿通过重新编码来完成这一工作。

但在有些情况下，在设计状态下对策略文件实现的配置选项只能满足支持应用程序所需的一般安全需求。而对于特定的安全需求，需在开发状态下用代码来实现。

3.5.1 在代码中定义策略

在 WSE 3.0 中，Policy 类对一个包含断言的策略建模。它有一个名为 Assertions 的公共属性（Collection<PolicyAssertion> 类型），用于管理一个 PolicyAssertion 实例的集合。有以下两种方式：

1. 通过实例化一个 Policy 对象并将 PolicyAssertion 对象添加到它的 Assertions 集合中，可以通过编程方式定义一个策略。

2. 也可以从 Policy 派生一个类，并在其构造函数中添加 PolicyAssertion 对象，而不是在每次需要时从头开始构建策略。

在代码中定义策略的方式，给动态构建策略提供了很大的运行时灵活性。可根据应用程序状态或配置设置选择是包括断言还是省略断言。

3.5.2 在 XML 文件中定义策略

WSE 3.0 提供了一个在 XML 中定义策略的内置解决方案。通过在配置文件中指定整个策略断言列表，以及动态加载断言类。允许在无需重新编译和重新部署代码的情况下对策略本身进行更改。

1. 策略文件

定义策略的配置文件被称为策略文件。是一个集合了被命名策略和策略扩充的 XML 文件。每个被命名策略用包含了一个有序的策略断言集的 <policy> 元素来声明，策略扩充用 <extension> 元素来声明。这些元素都是策略文件中<policies> 根元素的子元素。

下例显示了 WSE 策略文件的结构：

```
<policies>
  <extensions>
    <extension />
  </extensions>
  <policy name="ServerPolicy">
```

```
    ...
  </policy>
</policies>
```

每个<policy>使用名称属性指定各策略，以后可通过使用该名称在代码中引用策略。每个 <policy> 元素包含一个断言元素的列表，用来指定各策略断言。每个断言元素对应一个 PolicyAssertion 派生的类。在使用内置断言时，WSE 会自动对应，不需要指定。而对于自定义断言，就需要在<extensions>中，显式指定两者之间的映射。

在策略文件的<extensions>区中指定了处理各个断言的程序集信息。

2. 策略文件的配置

对于两端的策略文件的配置而言，其格式是一致的，而内容需要根据两端各自的安全需求有针对性地分别进行配置，通常具有对应性。

在 XML 文件中定义策略并且以一定格式进行配置，在灵活性和开发人员效率之间提供了最佳的平衡，这成为大多数开发人员所采用的方法。

3.5.3 将策略应用于服务

1. 在策略文件中定义并命名一个策略后，在 Web 服务 Web.config 文件中添加<policy>元素来指定包含了 Web 服务策略的策略文件。

包含策略的 XML 文件的架构允许声明任意数量的命名策略。这就允许同一个应用程序域中（因此共享一个配置文件）运行的不同服务能使用不同的策略。

2. 一旦 WSE 知道在何处查找基于 XML 的策略，只需要在服务类定义前设置属性指定策略名（如[Policy("ServerPolicy")]），就可使类中所有 Web 服务方法都应用该策略。

3. 在应用程序域中首次实例化服务类时，WSE 运行库将根据策略名解析为一个 Policy 实例，该实例在配置文件中引用的 XML 文件中声明。然后，该 Policy 实例将编译到一个包括传入消息和传出消息的 SoapFilters 的管道中。所有指向该 Web 服务的 SOAP 请求和响应将通过该管道得到处理。

3.5.4 将策略应用于客户端

1.给客户端应用程序的配置文件添加<policy>元素来指定包含了客户端应用程序策略的策略文件（注：该策略文件一般从Web服务端获得，确保所用策略与Web服务端所用的策略对应）。

2. 为了将策略应用于客户端上的服务代理。在实例化一个支持 WSE 的Web服务代理类后，就可用其SetPolicy 方法来指定所用策略。如：

```
ServiceWse serviceProxy = new ServiceWse();
serviceProxy.SetPolicy("ClientPolicy");
```

3. 若实现安全需求的安全凭证不能在策略文件中直接指定，则需要在代码中创建客户端的安全凭证，并在客户端应用程序代码中调用代理类中的SetClientCredential 方法来将该客户端凭证提供给代理类；然后再在客户端应用程序代码中用策略名调用代理类的SetPolicy方法。从而实现客户端策略的应用。

如：当使用用户名和密码作为安全凭证时，由于用户不同，无法在策略文件中直接指定，则需要通过代码实现：

```
ServiceWse proxy = new ServiceWse();
// 创建用户名令牌作为安全凭证
UsernameToken token = null;
string username = Environment.UserName;
string password = GetUsersPassword(username);
token = new UsernameToken(username, password);
//提供客户端凭证给代理类
proxy.SetClientCredential<UsernameToken>(token);
//指定策略
proxy.SetPolicy("ClientPolicy");
Console.WriteLine("returned:{0}", proxy.HelloWorld());
```

3.6 本章小结

在开发效率和扩展性方面，微软的WSE拥有无可置疑的优势。正是微软在开发工具和开发技术上提供的强力支持，使得企业应用产品能够快速和高效的开发和部署。WSE3.0的推出为实现Web服务安全提供了更加有效的工具，目前还处于不断发展之中，各种方法也还在不断探讨，因此需要进一步关注。

第4章 基于策略的 Web 服务安全方案的设计

4.1 安全模型需求分析

目前安全技术发展日新月异，使得应用程序的安全性需求也随之提升。为了提供可扩展性，Web 服务安全模型需要将当前可用的安全规范与技术和目前应用程序的安全性需求结合起来。通过将各种安全机制封装在一起，提供统一的安全接口，使安全的部署更灵活,更容易拓展新的安全机制，以适应现今日益增加的安全需求。

目前针对跨信任域的端到端通信的安全问题,还没有统一的解决方案，大多数 Web Services 都采用 SOAP 安全性扩展的方式，通过和 SSL /HTTPS、XML 加密技术与 XML 数字签名技术的结合，来实现跨信任域的端到端的安全通信。

4.1.1 WS-Security 安全模型

WS-Security 规范旨在提供一套可以用于构建一系列安全性协议的灵活的机制，自身并不保证安全性。虽然 WS-Security 通过扩展 SOAP 协议，为 SOAP 消息的加密、签名和携带安全性令牌提供了一个较好的解决方案。但并没有提供完整的安全性解决方案。需要与其它 Web 服务扩展和更高级的特定于应用程序的协议联合使用，以适应多种安全性模型和加密技术。

其体系结构如图 4-1 所示:

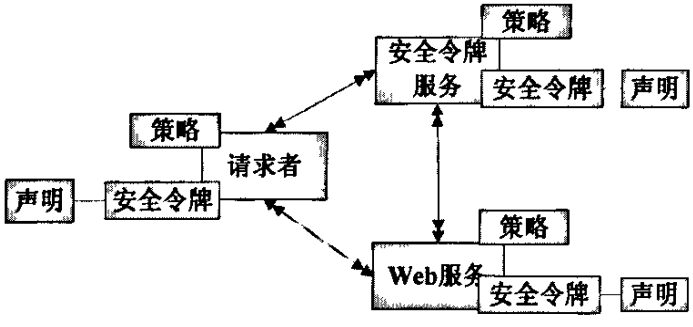


图 4-1 WS-Security 安全模型

安全模型中定义了两个概念:安全性令牌(Security Token)和Web服务端点策略(Endpoint Policy)。

安全性令牌表示一组与安全相关的信息集合。(例如，X.509 证书、用户名等)。安全性令牌可以采用PKI ,Kerberos或者其他的安全基础设施来实现。

Web 服务端点策略(Endpoint Policy)，由 Web 服务端根据自己的需要制定，要求所有服务请求者提供所需的声明(Claims)以及和这个声明相关的信息(如，姓名、密钥、授权等信息)。

其工作原理为：

1. 服务请求者向 Web 服务端发送请求消息的同时，必须根据 Web 服务端所制定的策略，提供相应的证书及声明，如果服务请求者无法提供，则 Web 服务端将拒绝此请求。

2. 为了得到相应的声明和证书，服务请求者或服务提供方与安全令牌服务（Security Token Service，STS）通信，以获得安全性令牌。

3. 当服务请求者具备相应的声明和证书之后，在请求服务时他们会将关联的安全性令牌随同 SOAP 消息一起发送给 Web 服务端，然后得到 Web 服务端的响应。

这个安全模型允许使用 X.509 证书、Kerberos 票据等技术，利用传输层和网络层中的安全措施，建立更高层次的密钥交换、验证、授权、审计和信任机制。

4.1.2 需求分析

（1）安全消息交换：Web 服务安全必须保证 SOAP 消息端到端的安全性，不仅要满足基本的安全需求，还要满足 Web 服务应用的一些特殊需求， 如消息的部分加 / 解密等。

（2）身份认证：对于企业级应用程序而言，Web 服务的数量将有很多，并且具有不同的安全需求等级和用户规模。对于大规模的用户群而言，应该应用简单的验证方式和快捷的加密算法。而对于数量较少的企业用户而言，就可以实施更为安全和全面的安全服务。

（3）授权：对于不同的用户 / 角色，Web 服务的可访问性是不同的。需要建立一个灵活、可扩展的访问控制模型[24]，对不同 Web 服务方法进行访问控制。

4.2 Web 服务安全解决方案的设计

本文在 WS-Security 规范提出的基本安全模型的基础上，针对 Web 服务的安全性需求，经过对安全 SOAP 消息交换、身份认证和访问控制三个 Web 服务安全性要素的深入研究，基于 WSE3.0 的策略模型，设计了一种基于策略架构的 Web 服务安全解决方案。

4.2.1 总体框架

该解决方案是一种面向服务的可信 Web 服务安全解决方案。如图 4-2 所示。

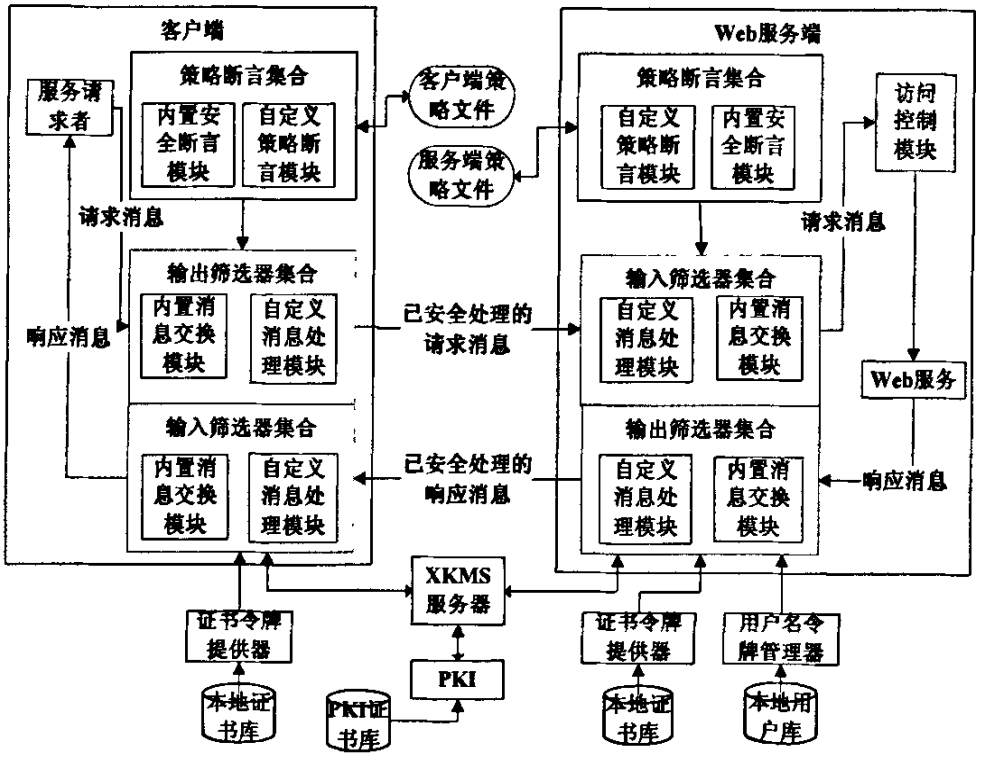


图 4-2 基于策略的 Web 服务安全解决方案的总体框架

在此安全解决方案中包括以下部分：

1. 策略

对 Web 服务实施有效管理，让每一个 Web 服务都应用一定的策略，来体现 Web 服务以及服务方法所需要的安全需求。由服务提供方根据实际需要编

写安全策略，并通过灵活配置来实现动态的分配和调整服务的安全等级、以及通过策略中的断言来动态创建和部署管道中的筛选器模块。

而服务请求者可以在执行安全消息交换之前，获取服务提供者所提供的安全策略文件，并设置与服务相应的安全配置，如采用的验证方式、哪个方法需要进行加密等。

2. 管道

由管道中的筛选器模块封装对 SOAP 消息的各种安全处理。根据安全策略中的断言，创建相应的筛选器模块，并根据策略断言所设的保护需求对消息进行相应的安全处理。如用户进行订单服务的策略中，就需要创建的筛选器模块提供特殊的加密处理，并要根据策略中断言的设定来决定是对消息的整个消息体进行加密，还是对其中的部分进行加密。这样方便了对 Web 服务安全功能的管理，并且提供了可扩展性和灵活性。

3. 其它辅助接口

解决方案的实现不能仅依靠策略的实现，必须要有一些其它接口提供支持。目前，在 WSE3.0 中，STS（安全令牌服务）还只能用于身份认证，对消息层安全保护的实现还在开发中。因此本方案采用了 X.509 证书来实现消息层的安全。由 XKMS（XML 密钥管理服务）服务器提供基于 XML 的公用密钥基础结构，用来管理密钥和证书信息，实现诸如密钥注册、验证、撤销等整套复杂的 PKI 系统功能。为服务请求者和服务提供者提供安全性令牌服务。

4. 其他辅助资源

解决方案中各功能的实现，需要有一些其他的信息和数据来辅助。如在用户验证中，需要证书库（用于少用户情况）和用户库（用于多用户情况）。

4.2.2 功能描述

客户端和服务端将根据安全策略中的断言形成断言模块，由断言模块创建相应的筛选器模块及进行筛选器模块的处理。

其他服务接口（如令牌提供器（提供令牌）等）则提供其他的相关数据供筛选器模块使用，为策略的实现提供支持。其他服务接口可能有一个或多个，也可能没有，这取决于功能的需要。

若根据策略断言创建的所有筛选器模块都处理完了，则可以认为当前的请求消息符合服务端所制定的安全策略，这时将该服务调用请求传递给服务

实现；否则将认为当前的请求非法，表示客户端无法满足服务端的安全需求来请求服务。同样，服务端也将根据该策略创建相应的筛选器模块来处理并响应来自客户端的请求。

各模块的功能如下：

1. 策略断言集合

根据应用的不同，可分为内置安全断言模块和自定义策略断言模块两种，内置安全断言模块只需提供所需的数据就可由 WSE3.0 自动处理，可以快速实现一般安全需求；自定义策略断言模块就需要自己实现处理，可以针对用户实现特定或复杂需求。

2. 筛选器集合

如上图 4-2 中虚框区域为筛选器模块。从图中可以看出，在 Web 服务端和客户端分别有一个输出筛选器集合和一个输入筛选器集合。集合中既可以包含由内置安全断言创建的消息交换模块（实现一组安全需求的整合模块），也可包含由自定义断言创建的组合的消息处理模块。如图 4-3 所示，包括基本安全功能子模块（如验证模块、加密/解密模块、签名/验证模块等）和高级安全功能子模块。也包括与安全相关的相应辅助功能模块，如跟踪日志模块等。

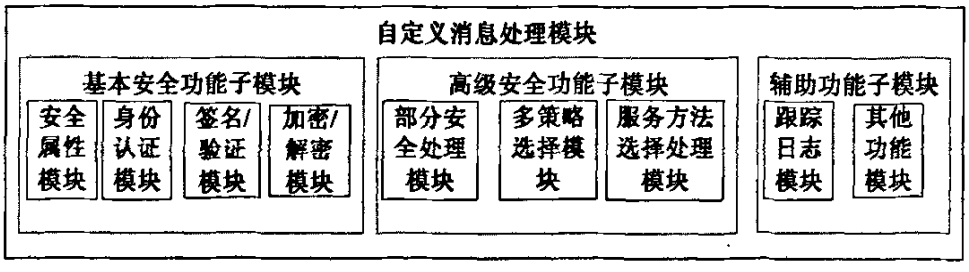


图 4-3 自定义消息处理模块图

筛选器集合由一对输入和输出筛选器集合组成。如在输出筛选器集合中包含了加密子模块，则在输入筛选器中就要包含解密子模块。

各个子模块的功能如下：

（1）安全属性模块

主要是通过在 SOAP 头中加入安全属性（如时间戳等信息）标头元素来满足用户额外的安全需求。

（2）身份认证模块

身份认证模块分客户端输出处理模块和服务端输入处理模块两部分。客

户端输出处理模块负责在请求Web服务时，将用户凭证添加到消息中。服务端输入处理模块负责接收用户凭证，并对客户端进行身份认证。

WS-Security 规范支持使用用户名令牌和二进制安全性令牌，以及自定义安全性令牌。

在不跨信任域时，采用用户名令牌进行直接验证，客户端需提供用户名和口令（或摘要形式）形成用户名令牌，并添加到消息中发送。对应的 Web 服务端应具备用户库来对接收到的用户名令牌进行直接验证。

对于跨信任域的验证，可采用二进制安全性令牌(如X.509证书)。通过可信第三方(如X.509 PKI或安全性令牌服务(STS,Security Token Service))签发安全性令牌代理不同信任域之间的信任，统一负责身份认证和令牌的签发和撤销。因此，客户端输出处理模块通过访问本地证书库，获取客户端证书（已由CA生成并保存在本地证书库中）生成证书令牌，或向STS申请，由STS 提供安全性令牌。服务端输入处理模块接收消息，从消息头中获取安全性令牌，验证其有效性，若是证书令牌，还需通过CA验证证书的信任链，再通过本地证书库（已从CA获得证书撤销信息）验证其撤销状态。

对于企业客户，一般采用证书验证方式，利用证书库来保存用户证书。对于数量很大的一般客户，则采用用户名/口令验证方式，将用户凭证保存在用户库中。

（3）加密/解密模块

加密模块属于输出处理模块，负责加密并格式化发送消息。解密模块属于输入处理模块，负责根据接收消息中的安全信息，将消息中的加密部分还原成明文。

本模块通过XML加密来实现机密性。具有部分加密的特性，可只对敏感内容加密。一般采用对称密钥与非对称密钥结合使用，并需要证书库提供相应证书支持。

（4）签名/验证模块

签名模块属于输出处理模块，负责对输出消息进行签名。验证模块属于输入处理模块，负责对请求消息进行签名验证。从而为消息提供数据完整性。

本模块通过 XML 签名来实现消息完整性。可采用对称密钥进行 MAC（消息身份认证）签名实现，也可采用非对称密钥进行数字签名实现。采用 MAC 签名只能实现消息的完整性，并不能保证是谁发送了该消息。而数字签名就能实现不可否认性。显然，也需要证书库提供相应证书支持。

3. 访问控制模块

访问控制模块属于 Web 服务端的输入处理辅助模块，其主要功能是采用基于角色的访问控制，控制用户对相应 Web 服务和服务方法的访问。如图 4-4 所示。首先由访问控制接口，通过身份认证模块获得用户的身份信息和其所要访问的目标 Web 服务等信息，然后根据用户角色关系表获得用户的角色信息，最后由访问授权模块根据授权策略获得该用户角色可以访问的 Web 服务列表。并判断目标 Web 服务是否在可访问服务列表中。若在则访问该 Web 服务，否则拒绝访问。

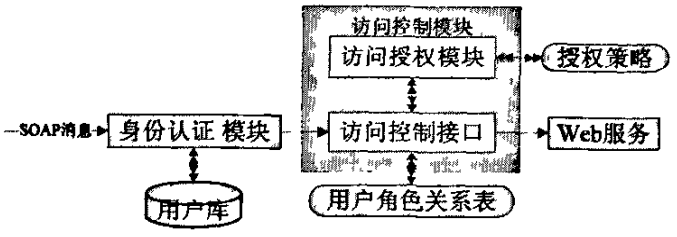


图 4-4 访问控制模块图

4.2.3 安全消息交换的处理流程

4.2.3.1 前期准备

（1）客户端的策略文件及服务端的策略文件。客户端应先获得服务端所提供的客户端策略文件。

（2）客户端的证书库（提供自身证书应带私钥）、服务端的证书库（提供自身证书带私钥）和用户库。服务端与客户端都已从本地证书库中获取各自的证书(带私钥)。并且客户端向 Web 服务端申请获得了 Web 服务端的证书。

4.2.3.2 服务端和客户端的配置

为实现安全消息交换，在客户端和服务端上都应该部署相应的配置，使得通信双方具备相互安全消息交换的能力。

1. 选择客户端验证方法是采用用户名\口令令牌还是采用 X.509 证书令牌，并指定其具体内容。

2. 指定 SOAP 消息所需要的保护（完整性和机密性），并指定加密和数字

签名所需证书。如：可指定SOAP消息必须被签名并且SOAP消息体和签名必须被加密，因此需要先签名后加密。

4.2.3.3 安全消息交换的流程描述

（1）客户端读取客户端策略文件，对所用策略进行解析，对策略元素中的断言元素依次根据注册的策略扩展形成断言，并依次执行各个断言，若为自定义断言，则需自己根据断言元素内容进行相应解析和处理，对于自定义断言元素的内部子元素根据该断言的策略扩展集依次形成断言，并以类似方式进行自行处理。同时获得相应属性值，以后提供给对应的筛选器处理消息时使用；若为内置断言，则会自动进行解析和处理，创建内置消息交换模块。由各断言依次创建相应的输出筛选器(同时获取令牌)，形成输出筛选器集合。

（2）由各个输出筛选器依次根据属性对请求消息进行安全处理（包括建立验证、签名和加密等）。并将处理后的请求消息发送给服务端。

（3）服务端接收到请求消息后，同样读取服务端策略文件并对所用策略进行解析，对断言元素依次根据注册的策略扩展形成断言，并依次执行各个断言，处理方式与客户端一致。由各断言依次创建相应的输入筛选器（同时获取令牌），形成输入筛选器集合。

（4）由各个输入筛选器依次根据属性对请求消息进行安全处理（包括用户验证、签名验证和解密等），最终获得原始请求消息。如图4-5所示。

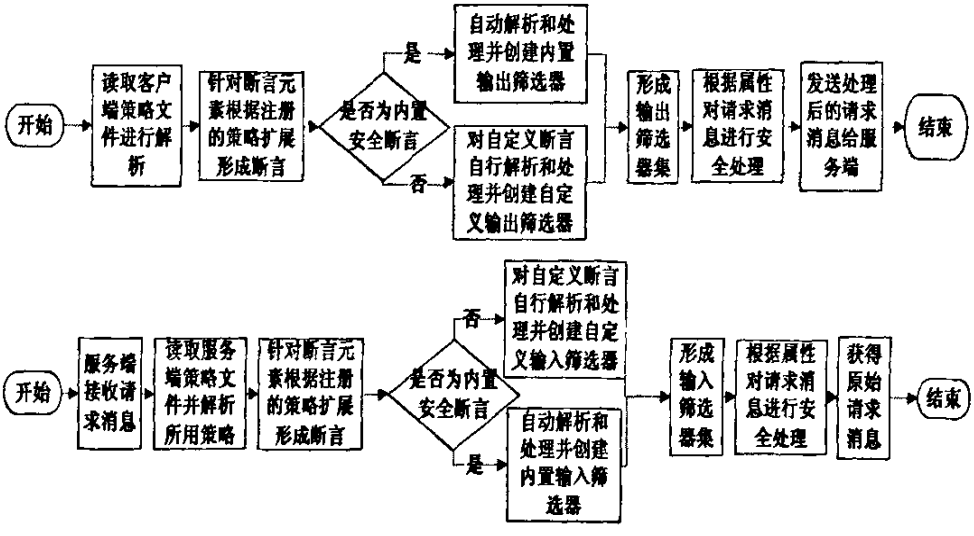


图4-5 请求消息安全交换流程图

至于服务端将响应消息发送给客户端，而客户端接收响应消息，其中的安全处理过程与请求消息的处理是类似的。

上述消息交换过程都设有超时机制， 若超时则重新执行整个通信过程。若在消息交换过程中发生错误，例如，验证未通过，解密、验证失败等，则终止通信，发出错误响应消息。

4.2.3.4 安全处理的流程描述

下面详细论述在安全 SOAP 消息交换中，筛选器实现安全处理的工作流程。

1．应用用户名/口令验证方式的消息安全处理

在 WS-Security1.1 中支持使用对称密钥签名和加密。可采用由客户端创建对称密钥来签名和加密数据，由此提高处理效率。为了让服务端也获得该对称密钥，客户端需利用非对称密钥加密该对称密钥再传送给服务端。EncrptedKey 是实现此种功能的扩展元素。

因此，安全处理的实现流程为：

（1）生成要发送给 Web 服务端的请求消息，在 SOAP 头中添加安全属性（如时间戳等信息）来防止对 Web 服务的重放攻击，必要时可根据用户额外的安全需求对安全属性进行相应的扩展。

（2）客户端根据服务端指定的身份认证方式，构造相应的身份认证安全性令牌(这里是用户名/口令方式，则将用户名/口令形成用户名令牌；若是证书方式则从本地证书库中获取客户端的证书，并用该证书形成证书令牌)，并将身份认证安全性令牌附加到 SOAP 头中。

（3）客户端创建一个随机的对称密钥。（该对称密钥将用于根据安全需求对消息中要求完整性和保密性的部分进行签名和加密。）

（4）从本地证书库中获取服务端的证书，并将该证书形成证书令牌，用其中的公钥加密该对称密钥，保证只有服务端能够获得该对称密钥。以 xenc:EncryptedKey 元素的形式将对称密钥序列化到 Security 标头中。

（5）用该对称密钥对包含代表客户端凭据的用户名令牌加密，形成 xenc:EncryptedData 元素，并由 xenc:EncryptedKey 元素内部的引用列表指向。这样，客户端的密码就不会暴露在网络上了。如图 4-7 所示，SOAP Head 中的 wsse:UsernameToken 已经被加密，被 xenc:EncryptedData 所替代，其

Token Reference 表示加密使用的 Key 来自 xenc:EncryptedKey。

(6)用对称密钥根据签名需求对要求完整性的所有消息部分、时间戳以及 xenc:EncryptedData 中加密的纯文本 wsse:UsernameToken 元素进行 MAC 签名。形成请求消息中 Security 标头中的最后一个元素 dsig:Signature。

(7) 用该对称密钥根据加密需求对 SOAP 消息中要求机密性的所有消息部分进行加密。然后就可以将处理好的请求消息发送了。

(8) 注意：服务端的 X.509 证书不会出现在请求消息的 Security 标头中。由于客户端和服务端都可以通过各自的证书库访问它，因此不需要。相反，使用了一个描述该证书属性的外部引用。

(9) 服务端接收后，从本地证书库中获取服务端证书（带私钥）。并用该证书中的私钥解密获得对称密钥。

(10) 用对称密钥解密用户名令牌，同从用户库中获取的用户口令信息进行比较，进行用户验证。

(11) 用对称密钥解密消息中其他的被加密元素，达到实现机密性的目的。

(12) 用对称密钥验证 MAC 签名，达到验证消息完整性的目的（注：MAC 签名无法实现不可抵赖性）。

(13) 检查 SOAP 头中的安全属性，判断是否为重放、是否处于有效期内及是否为相应的发送者和接收者。

(14) 若所有验证通过，则服务端提供服务，生成响应消息。

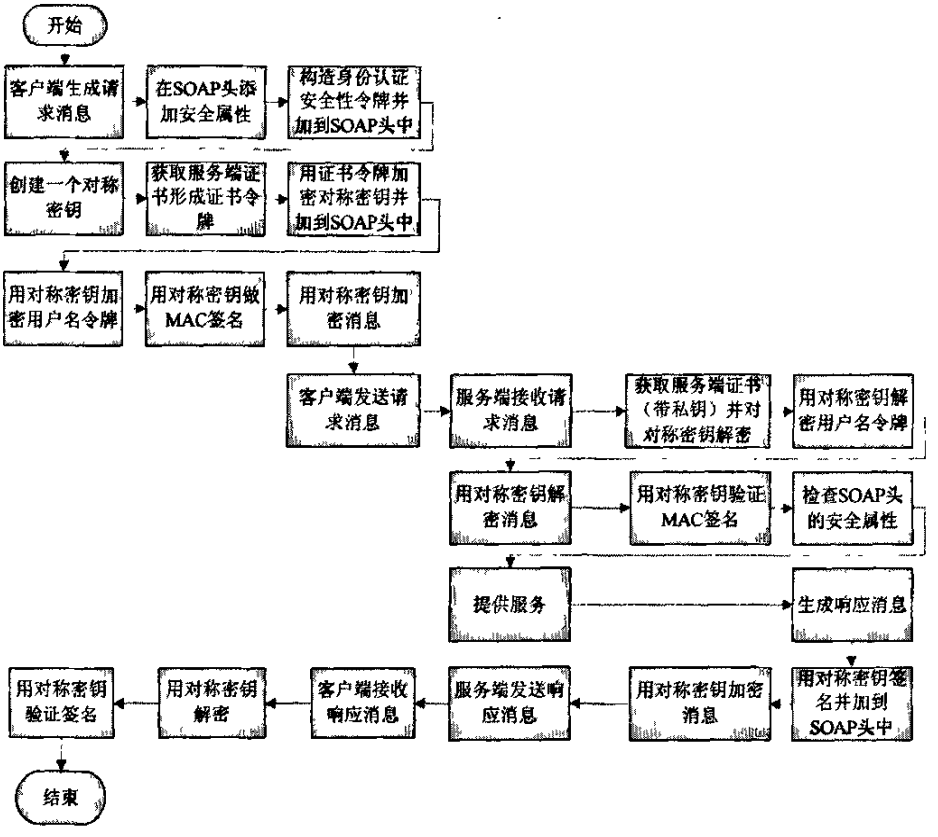
(15)服务端用客户端在请求中传递过来的对称密钥对 SOAP 消息中要求完整性保护的消息部分(包括时间戳元素和其他需要的元素)进行签名，并将签名信息放入消息头中。

(16) 用该对称密钥对 SOAP 消息中要求机密性保护的消息部分(这时不存在用户名令牌元素)进行加密，并由 xenc:ReferenceList 指向加密消息部分。然后发送给客户端。

(17)客户端用自己的对称密钥根据 xenc:ReferenceList 对加密消息部分进行解密，从而实现响应消息的机密性。

(18) 客户端接收到响应消息后，用自己的对称密钥对签名信息进行验证，由于对称密钥只能被服务端获取并用来签名，因此实现了客户端对服务端的身份认证。

处理流程图如图 4-6 所示。



图

图 4-6 应用用户名/口令验证方式的消息安全处理流程图

下图 4-7 就是已经实现安全处理后的请求消息与响应消息的 SOAP Envelope 扩展结构。

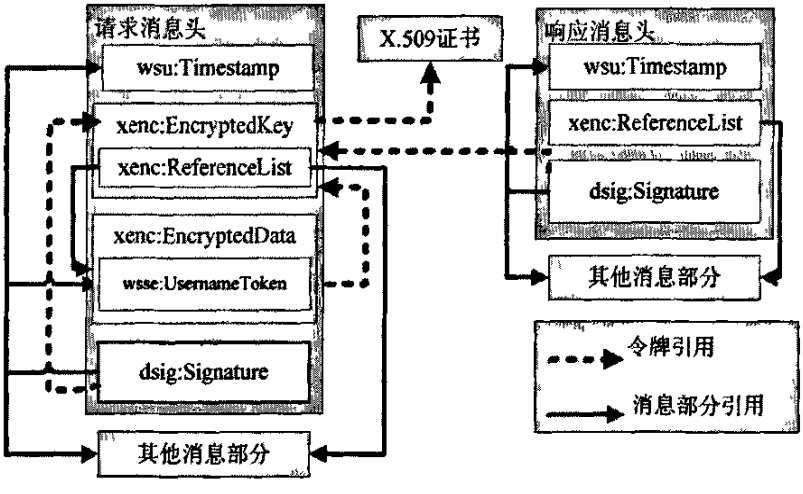


图 4-7 显示请求和响应消息的 Security 标头的布局示例

在Security Header中包含的部分有：(1) 时间戳 (防重放攻击)(2)用服务端证书的公钥加密的对称会话密钥（提供给对方解密） (3)用对称会话密钥加密的用户名令牌 (提供身份认证)(4)用对称密钥对时间戳、用户名令牌和其他要签名部分所做的MAC签名（提供消息的完整性）

根据完整性和保密性要求，以及断言配置，格式可能会有所不同。特别是，如果更改对消息应用签名和加密的顺序，布局将有所不同。默认情况下，先对消息进行签名，然后再进行加密。

2. 应用证书验证方式的消息安全处理

使用用户名/口令验证方式一般用于保证个人用户和企业之间的安全，而企业与企业之间常使用证书验证方式来保证两者之间的安全。一方面因为使用X.509证书的安全性要高于用户名/口令，另一方面从现实的角度，不可能为每个企业分配一个用户名/口令。

在采用两个X.509证书来实现相互验证中，加入用户的X.509证书的原因是要利用用户的私钥来签名消息，从而让服务端通过证书来鉴别用户的身份。于是在SOAP Head中会出现两个X.509证书。

因此，请求消息安全处理的实现流程为：

(1)客户端创建对称密钥，并根据安全需求对请求消息中需要进行安全保护的部分进行签名/加密处理：使用客户端的私钥对用对称密钥做的MAC签名进行数字签名，并将签名和客户端证书添加到消息中；使用服务端的公钥对对称密钥进行加密。最后将处理后的消息发送给服务端。

(2)服务端收到请求消息后，先提取消息中客户端的证书，验证其有效性，再利用PKI验证证书的信任链，再到PKI证书库验证证书的撤销状况；通过后，利用证书所提供的公钥对MAC签名所做的数字签名进行验证，实现不可抵赖性；用自己的私钥解密对称密钥，用该对称密钥验证MAC签名，实现消息的完整性；用该对称密钥解密消息加密部分，实现消息的机密性。检查SOAP消息头中的安全属性，判断是否为重放、是否处于有效期内及是否为相应的发送者和接收者。

至于响应消息的安全处理过程与用户名验证方式是一致的，就不再论述。

不直接用私钥对消息进行数字签名，而是用对称密钥进行MAC签名，再用私钥对MAC签名进行数字签名，是由于利用摘要(SHA)一方面可以减少数据传输量，另一方面可以增强安全性。

4.2.4 方案的优点与不足

优点有：

（1）该方案效率高，加密/解密速度快。对于大量XML数据的传输采用了对称密钥进行加密和解密，充分发挥了其加解密效率高、速度快的特点；而对于验证信息和会话密钥等少量的数据采用非对称密钥进行加/解密操作。

（2）消息交换双方尽量减少在消息中携带密钥，所携带密钥也是经过加密，减小了密钥泄漏的风险。

（3）能提供多种身份认证方式。服务端通过身份认证令牌来对客户端进行身份认证。

（4）密钥管理方便，虽使用了对称密钥，但不需保存该密钥，在消息交换连接建立之前产生，消息交换结束后消去。

（5）能实现对SOAP消息的部分内容进行安全处理。

需要注意和考虑的问题有：

（1）部署服务时各属性的配置不能出错。

（2）安全模块虽可以独立创建。但是其灵活性还需提高，如部分加密针对具体应用还需要进一步考虑。

（3）对一个实际系统而言，证书的发布管理需要继续考虑。

（4）安全性带来的系统性能的损失是需要注意的。

4.3 本章小结

本章分析了基于策略的Web服务安全解决方案的框架设计，对各模块的功能进行了详细讨论。该方案将安全策略的制定与策略的实现机制进行了分离，既方便策略的修改，也方便策略实现机制的升级。服务端可以方便灵活地选择和变更安全策略，而无需关注安全策略的实现细节，从而实现了更加安全高效的Web服务。

第5章 基于策略的Web服务安全方案的实现

5.1 应用系统的实现

5.1.1 系统描述

为了验证前章所设计解决方案的安全性，本人初步开发了一个常用的电子商务中的Web服务应用系统，来提供Web服务安全实现所需的环境。

该系统为网上销售系统，由eBusiness公司作为Web服务端，向用户提供产品服务。用户利用客户端应用程序作为客户端，访问eBusiness提供的服务。用户可通过两种方法来访问，一种是为一般客户，通过提供用户名/口令来访问。另一种是代表企业客户，通过提供X.509证书来访问。eBusiness向用户提供Web服务，需要维护所有用户的身份认证信息。所提供业务主要包括查询、支付、客户管理等基本功能。在设计上，采用了.net petshop 4.0中的设计模式 。

5.1.2 开发平台和开发语言的选择

在本文中，采用Visual Studio 2005用于生成Web服务端应用程序和客户端应用程序，形成一个.net平台下的分布式应用系统。而Web服务的开发在其中属于中间件的开发，由Web服务端应用程序提供。并由客户端应用程序提供用户界面，实现对Web服务的引用。另外，采用WSE3.0实现Web服务两端的消息交换安全。

5.1.3 系统分析

1. 系统的体系结构

客户端应用程序主要包括：表示层、业务逻辑层、Web服务引用层、Web服务安全层。

服务端应用程序主要包括：Web 服务层、业务逻辑层、数据访问层、数据层、Web服务安全层。

其简要体系结构如图 5-1 所示：

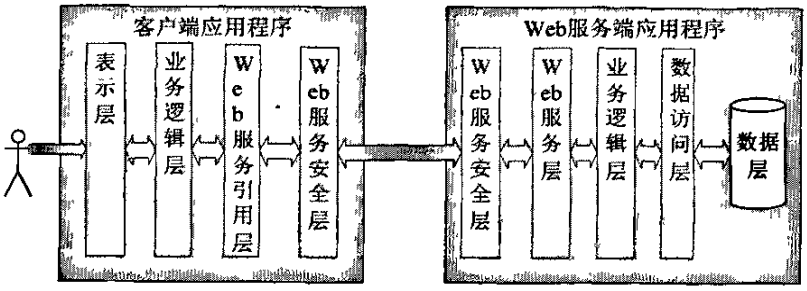


图 5-1 系统的简要体系结构图

客户端应用程序的项目如图 5-2 所示：

图 5-2 客户端应用程序的项目图

- 表示层：提供与用户交互的界面和界面处理流程。由 WebUI 项目实现。
- 业务逻辑层：提供客户端应用程序所需的业务逻辑，具体业务通过 Web 引用层向相应的 Web 服务端应用程序请求。由 BLL 项目实现。
- Web 引用层：提供对 Web 服务端应用程序中 Web 服务的引用，形成 SOAP 消息。通过 WebUI 项目中的 App_WebReference 实现。
- Web 服务安全层：提供该客户端所需的 Web 服务安全支持。形成安全 SOAP 消息。除了用 WSE3.0 提供基本支持外，还由 CustomPolicyAssertionLibrary 项目及 SecurityPolicyBase 项目提供特殊安全支持和辅助安全支持。

Web 服务端应用程序的项目如图 5-3 所示：

图 5-3 Web 服务端应用程序的项目图

- Web 服务安全层：提供 Web 服务端所需的 Web 服务安全支持。处理安全 SOAP 消息。除了用 WSE3.0 提供基本支持外，还由 CustomPolicyAssertionLibrary 项目及 SecurityPolicyBase 项目提供特殊安全支持和辅助安全支持。
- Web 服务层：为客户端应用程序提供服务接口，启动 Web 服务，负责处理 SOAP 消息。由 WebService 项目实现。
- 业务逻辑层：负责处理业务请求，实现 Web 服务端具体的业务逻辑。由 BLL 项目实现。
- 数据访问层：实现对数据层的访问，抽象众多的后端系统和数据源。由 SQLSERVERDAL 项目实现。
- 数据层：存储 Web 服务端应用程序的数据，在该系统中采用 SQL SERVER2005 数据库。

2. 系统的设计思想

作为电子商务的应用，常用业务功能一般是一致的，而本论文的主要目标不是在设计应用平台上，而是设计在此基础上的 Web 服务安全。所以对具体业务服务不做详细设计，而只立足一般常用功能。主要关注客户端与 Web 服务端之间的业务服务的安全消息交换的实现，及满足特殊业务服务的特殊需求。如订单服务中的部分加密需求。由此逐步建立各个安全功能模块。

WSE3.0 简化了安全模型，它在消息交换上的整合使用替代了组合独立的安全功能模块的使用。因此，在应用程序安全系统设计时，应通盘考虑系统内整体的安全需求，设计相应的策略文件，组合服务两端的安全机制实施整体安全部署。

Web 服务安全策略基于业务要求。根据安全需求的不同，将业务服务组

件分为一般业务服务组件和特殊业务服务组件，一般业务服务组件采用一般策略保证安全，而特殊业务服务组件采用自定义策略实施更高要求的安全。

5.1.4 系统安全需求的实现方法

整体框架设计的基本思路就是先围绕基本的功能业务需求，从一个客户端和一个服务端开始设计。以后就可逐步根据功能的添加再扩大规模，给平台添加新的业务功能组件，从而也带入新的安全需求。

1. 身份认证需求的实现方法

服务端的安全职责之一是面向客户的，须维护与客户有关的信息，负责对客户进行身份认证。

（1） 客户首先要在Web服务端进行注册

客户端应用程序作为客户，若想实现对Web服务端应用程序中的Web服务进行引用，则首先需要在Web服务端应用程序中进行注册。

若为一般客户，则采用用户名/口令方式注册，并将客户信息保存在客户库中。若为企业客户，则采用证书方式注册，除了保存客户信息在客户库中外，还将客户端证书保存在证书库中。

通过注册，服务端一旦认同，则向客户端提供服务端证书，并提供适用于该客户端的安全策略文件及对应的Web服务安全程序集。客户端应用程序就可建立起与该Web服务端应用程序对应的Web服务安全层。

（2） 服务端必须确定请求发送方是谁。

一旦客户端应用程序因用户业务操作而需要通过Web服务引用层与Web服务端应用程序联系，请求相应的Web服务。则需要客户端应用程序在请求消息中提供客户端令牌给Web服务端应用程序，与注册时客户端提供的用户名/口令或证书比较进行客户端身份认证。若请求客户是一个无效客户，则不提供响应信息。如对于企业客户，eBusiness在它提供出Web服务后能从使用Web服务的企业处收取相应的费用。可通过在eBusiness使用验证企业客户签名实现该需求。

双方还需要确定消息中所用证书的有效性及是否未被撤消，还有证书的签名者是否可信任。这属于PKI的一部分。

（3）客户端必须确定响应发送者是谁。

可通过在客户端上对响应消息验证签名来自eBusiness来实现该需求。

2. 安全消息交换需求的实现方法

（1）机密性：在客户端与服务端之间交换的有些是需要保密的数据，如报价、帐号等。

（2）完整性：可通过消息签名模块来处理。如对于企业客户，在发送购买消息时，对包含eBusiness报价的消息签名。该签名表明客户同意支付这些商品的款项。

在客户端与Web服务端整个通信过程中，将通过两端的Web服务安全层的对应设置保障消息的完整性和机密性。

3. 授权需求的实现方法

应建立一个简单的基于角色的访问控制策略来简化对客户的管理。可根据具体业务情况，在eBusiness中建立相应的授权策略和用户－角色关系表，只允许客户访问各自特定的授权服务。因为eBusiness提供了很多服务，但该客户只签约了使用某一部分。所以要限制客户的使用。如一般客户由于所提供的用户名令牌无法进行数字签名，不具备不可抵赖性，因此也就不具备访问支付服务的权限。

WSE提供了内置的基于角色安全的授权功能，与Windows平台支持密切结合。能通过使用策略来进行声明设置，并且以后策略能应用于角色中的每一个用户。

5.2 基本安全需求的实现

WSE3.0通过内置安全断言整合了安全消息交换中各基本模块的功能实现。很多操作可以在此基础上自动完成，节省了大量的基本操作的开发时间。通过内置安全断言的使用，可以快速构建Web服务安全通道。

5.2.1 内置安全断言的应用

本系统根据应用环境及客户情况，采用以下两种内置安全断言来建立基本的安全通道。

- usernameForCertificateSecurity：针对一般客户，（1）身份认证：客户端用用户名和口令确认用户身份（UsernameToken），服务端用X.509证书确认（X509SecurityToken）。（2）安全消息交换：用

X509SecurityToken 安全令牌实现消息级安全。

- mutualCertificate11：针对企业客户，（1）身份认证：客户端和服务端都用 X.509 证书确认。(2)安全消息交换：用 X509SecurityToken 安全令牌实现消息级安全。需要支持 WS-Security 1.1。

1. 主要的相关类

（1） 令牌类：

SecurityToken：安全令牌类，常作为基类。

X509SecurityToken：对应 X.509 证书形成的令牌。

UsernameToken：对应用户名/口令形式形成的令牌。一般用作客户端的身份认证。

（2） 令牌相关类：

usernameTokenManager：用户名令牌管理器类，通过重载 AuthenticateToken 方法，用令牌中的用户名来从用户库中获得口令返回，提供给 WSE 来同令牌中的口令比较。

x509TokenProvider：X.509 证书令牌提供器类，通过调用 CreateToken 方法从证书存储区中获得证书信息并创建令牌。

X509SecurityTokenManager：X.509 证书令牌管理器类，通过调用 VerifyToken 方法实现对 X.509 证书令牌的验证。

（3） 签名类：

MessageSignature：实现对签名数据的处理。

（4） 加密类

EncryptedData ：实现对加密数据的处理。

2. 主要的相关数据文件

（1）应用程序配置文件

如下所示为 Web 服务端中的 Web.config 文件中的部分相关配置信息

```
<microsoft.web.services3>
  <security>
    <x509 allowTestRoot="true"/>
    <!-- 增加一个自定义安全令牌管理器到安全令牌管理器集中，用来处理接受到的用户凭证，
    UsernameToken作为消息中不带命名空间前缀的令牌类型 -->
    <securityTokenManager>
        <add type="Microsoft.Web.Services3.WebServicefyj.CustomUsernameTokenManager,
```

```
            SecurityPolicyBase" namespace="…" localName="UsernameToken"/>
        </securityTokenManager>
    </security>
    <!—指定所用的策略文件 —>
    <policy fileName="wse3policyCache.config"/>
</microsoft.web.services3>
```

（2）策略文件

由于将 WS-Policy 用作配置策略的语法对大多数开发人员而言过于复杂和麻烦。WSE 3.0 定义了自己的词汇表来表示策略和内置断言，而未使用 WS-Policy 定义的词汇表。使产生的 XML 更干净且更易于使用。

如下所示为客户端的 wse3policyCache.config 策略文件中关于应用 usernameForCertificateSecurity 内置断言策略的部分信息。

```
<policies xmlns="http://schemas.microsoft.com/wse/2005/06/policy">
    <extensions>
        <extension name="usernameForCertificateSecurity"
        type="Microsoft.Web.Services3.Design.UsernameForCertificateAssertion, …" />
        …
    </extensions>
    <policy name="ClientPolicyfyj">
        <usernameForCertificateSecurity …>
            <clientToken>
                <username username="administrator" password="ffyyjjok" />
            </clientToken>
            <serviceToken>
                <x509 storeLocation="CurrentUser" storeName="AddressBook" findValue="CN=ffyj"
                findType="FindBySubjectDistinguishedName" />
            </serviceToken>
            <protection>
                …
            </protection>
        </usernameForCertificateSecurity>
        <requireActionHeader />
```

```
</policy>
...
</policies>
```

对于各个具体的基本功能模块的实现，由于在自定义功能实现中还需要应用到，所以这里只作简单叙述。

5.2.2 身份认证模块的实现

采用用户名安全令牌实现身份认证。包括提供身份信息的客户端模块和验证身份信息的Web服务端模块。

1. 客户端模块

用户名安全令牌由客户端生成，并设置给服务代理类使用。实现步骤为：

（1）获取当前系统用户名，并获得相应口令。

（2）根据令牌调用标志判断，若允许调用，则创建一个带正确口令的用户名令牌实例，否则创建一个带错误口令的用户名令牌。

（3）将令牌设置为服务代理类对象的客户端安全凭证。

2. Web服务端模块

为了在Web服务端验证用户身份，通过自定义安全令牌管理器获取在Web服务端的用户库中所保存的用户口令，再提供给WSE与用户名安全令牌中的口令比较，进行自动验证。实现步骤为：

(1)在服务端创建一个从UsernameTokenManager类中派生的自定义用户名令牌管理器，负责验证接收到的消息中的安全令牌。

(2)在Web.Config文件中添加相应配置<securityTokenManager>元素，指定自定义安全令牌管理器的位置。

（3）重载UsernameTokenManager类中的AuthenticateToken方法来处理身份认证。

（4）在该方法中，根据提供的用户名安全令牌中的用户名，访问用户库获得用户口令并返回；

（5）提供给WSE进行自动验证。若一致，则通过。

5.2.3 数字签名模块的实现

能用于数字签名的安全令牌类型有：X.509 证书、用户名与口令令牌

(UsernameToken)、Kerberos 票据安全上下文令牌、自定义安全令牌(如 SAML 令牌)。本系统采用 X.509 证书对 SOAP 消息签名，发送者需要通过使用自己的 X.509 证书中的私钥签名 SOAP 消息。而接收方通过发送者 X.509 证书中的公钥来验证该签名，判断该消息是否来自发送者。因此，为实现签名，发送者必须能访问自己的 X.509 证书和私钥，而接收者必须能访问发送者的 X.509 证书。

具体实现由自定义策略断言创建的输出自定义筛选器中的 SecureMessage 方法完成。默认情况下，WSE 对<Body> 元素的整个内容，Security 头的 <Timestamp>元素，和所有寻址标头一起进行签名。

实现步骤为：

(1) 创建自定义策略断言。

(2)创建自定义输出筛选器，重载 SecureMessage 方法，实现以下操作。

(3) 在 SecureMessage 方法内，获取要签名 SOAP 消息的安全令牌。

客户端须先获得客户端带有私钥的 X.509 证书，通过调用 GetSecurityToken 方法实现，在当前用户的证书存储区根据主题名称搜索一个指定的 X.509 证书。如：

```
X509SecurityToken signatureToken = GetSecurityToken("CN=fyjClient");
```

(4) 加到 SOAP 头中。如：

```
security.Tokens.Add(signatureToken);
```

(5) 使用指定安全令牌创建一个 MessageSignature 类的新实例，实现签名；并加到 WS-Security SOAP 头中。

如：

```
MessageSignature sig = new MessageSignature(signatureToken);
security.Elements.Add(sig);
```

由于签名消息没有加密，仍是 XML 格式的明文文本，在传输之前一般要先进行加密。

5.2.4 加密模块的实现

能用于加密 SOAP 消息的安全令牌类型与签名令牌类型一致。并且也是通过自定义策略断言创建的输出自定义筛选器中的 SecureMessage 方法实现。默认情况下，发送者对<Body>元素的整个内容而不对 SOAP 消息头进行加密。

实现步骤为：

(1) 创建一个自定义策略断言。

(2)创建自定义输出筛选器,重载 SecureMessage 方法，实现以下操作。

(3) 在 SecureMessage 方法内，获取要加密 SOAP 消息的安全令牌。

客户端须先获得 Web 服务端的 X.509 证书，也是通过调用 GetSecurityToken 方法实现，在当前用户的证书存储区根据主题名称搜索一个指定的 X.509 证书。如：X509SecurityToken encryptionToken = GetSecurityToken("CN=fyjServer");

(4) 用该安全令牌创建一个 EncryptedData 类的新实例，实现加密；并加到 WS-Security SOAP 头中。

如：EncryptedData enc = new EncryptedData((encryptionToken);

security.Elements.Add(enc);

5.2.5 验证模块的实现

1. 验证签名模块

一旦 WSE 在接收方配置，WSE 将根据配置自动获取相关信息来验证签名。但若想确定签名是否存在和签名是否应用到了预期的一组 XML 元素中，则需要通过代码来实现。实现步骤为：

(1) 创建一个自定义策略断言及相应的输入筛选器。

(2) 在客户端或 Web 服务端的输入筛选器中，重载 ValidateMessageSecurity 方法。

(3) 在该方法中，遍历 security 参数的 Elements 集。确定实现 ISecurityElement 接口的对象为 MessageSignature 类的一个实例，验证预期的 XML 元素是用指定的安全令牌签名。

2. 验证加密模块

一旦 WSE 在接收方配置，WSE 将根据配置自动获取相关信息来解密。但若想确定加密是否存在和加密是否应用到了预期的一组 XML 元素中，则需要通过代码来实现。实现步骤为：

(1) 创建一个自定义策略断言及相应的输入筛选器。

(2) 在客户端或 Web 服务端的输入筛选器中，重载 ValidateMessageSecurity 方法。

(3) 在该方法中，遍历 security 参数的 Elements 集。确定实现 ISecurityElement 接口的对象是 EncryptedData 类的一个实例，验证预期

的XML元素是用指定的安全令牌加密。

5.2.6 基本安全需求的实现结果

在此，由客户端提供客户端证书形成令牌，提供给Web服务端实现身份认证，并用该客户端证书进行签名。同时用服务端证书公钥实现加密。处理前后的SOAP消息对比如下所示。

处理前的消息简要格式为：

```
<soap:Envelope…>
    <soap:Header>
      <wsa:Action …>http://…/fyj/GetProductsByCategory</wsa:Action>
      …
      <wsa:To …>http://localhost:1472/WebService/Service.asmx</wsa:To>
    </soap:Header>
    <soap:Body wsu:Id="Id-ff25cb5a-936d-4ade-a446-e51c0f792738">
      <GetProductsByCategory xmlns="http://tempuri.org/fyj">
        <category>BIRDS</category>
      </GetProductsByCategory>
    </soap:Body>
  </soap:Envelope>
```

处理后的消息简要格式为：

```
<soap:Envelope…>
    <soap:Header>
     <wsa:Action …>http://…/fyj/GetProductsByCategory</wsa:Action>
      …
     <wsa:To …>http://localhost:1472/WebService/Service.asmx</wsa:To>
     <wsse:Security soap:mustUnderstand="1">
     <wsu:Timestamp wsu:Id="Timestamp-fea70cc1-a747-4daa-86e0-dbda73cd83a1">
          <wsu:Created>2006-10-20T02:39:18Z</wsu:Created>
          <wsu:Expires>2006-10-20T02:44:18Z</wsu:Expires>
     </wsu:Timestamp>
 //服务端证书公钥，用来加密对称会话密钥和验证签名公钥（客户端证书公钥）
  <xenc:EncryptedKey Id="SecurityToken-03cb7a67-7102-46a2-ada5-bcd86f36df79" …>
```

```
    ...
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
     <wsse:SecurityTokenReference>
<wsse:KeyIdentifier ...>PK9f914kfJ8xoYC7aKA+nVQc+e4=</wsse:KeyIdentifier>
     </wsse:SecurityTokenReference>
    </KeyInfo>
   <xenc:CipherData>
     <xenc:CipherValue>...</xenc:CipherValue>
   </xenc:CipherData>
 </xenc:EncryptedKey>
//用于加密数据的对称会话密钥
  <wssc:DerivedKeyToken wsu:Id="SecurityToken-5b9c3c41-e049-4169-93ba-262d9686c89c" ...>
  <wsse:SecurityTokenReference>
    //加密所用的服务端证书公钥的引用位置
    <wsse:Reference URI="#SecurityToken-03cb7a67-7102-46a2-ada5-bcd86f36df79" ... />
  </wsse:SecurityTokenReference>
   ...
 </wssc:DerivedKeyToken>
//加密数据的引用位置
  <xenc:ReferenceList xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <xenc:DataReference URI="#Enc-ccb561fb-9b69-4650-b114-c2bb74dc71ab" />
  </xenc:ReferenceList>
 //加密数据之一
 <xenc:EncryptedData Id="Enc-ccb561fb-9b69-4650-b114-c2bb74dc71ab" ...>
    <xenc:EncryptionMethod ... />
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
     <wsse:SecurityTokenReference>
     //加密所用对称会话密钥的引用位置
       <wsse:Reference URI="#SecurityToken-5b9c3c41-e049-4169-93ba-262d9686c89c" .../>
     </wsse:SecurityTokenReference>
    </KeyInfo>
    <xenc:CipherData>
```

```
        <xenc:CipherValue>…</xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedData>
  //客户端证书公钥，提供给服务端验证签名时所用
  <wssc:DerivedKeyToken wsu:Id="SecurityToken-553320f9-b2d0-4a11-81ea-c52f8b46e7c3" …>
      <wsse:SecurityTokenReference>
     //加密所用的服务端证书公钥的引用位置
       <wsse:Reference URI="#SecurityToken-03cb7a67-7102-46a2-ada5-bcd86f36df79" … />
      </wsse:SecurityTokenReference>
      …
  </wssc:DerivedKeyToken>
  //签名数据
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
          <SignedInfo>
            <ds:CanonicalizationMethod … />
            <SignatureMethod … />
            …//其他签名部分
    //时间戳部分
    <Reference URI="#Timestamp-fea70cc1-a747-4daa-86e0-dbda73cd83a1"> …
    </Reference>
    <Reference URI="#Id-ff25cb5a-936d-4ade-a446-e51c0f792738">//消息体部分
      <Transforms>
        <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
      </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
           <DigestValue>H7bSZTBWe39IkZ6fcxbYdtXD3wY=</DigestValue>
     </Reference>
    </SignedInfo>
    <SignatureValue>b4EqxmH8MpdFy05iECp58NXH1Ac=</SignatureValue>
      <KeyInfo>
        <wsse:SecurityTokenReference>
      //签名所用客户端证书公钥的引用位置
```

```
        <wsse:Reference URI="#SecurityToken-553320f9-b2d0-4a11-81ea-c52f8b46e7c3" … />
       </wsse:SecurityTokenReference>
      </KeyInfo>
    </Signature>
   </wsse:Security>
 </soap:Header>
<soap:Body wsu:Id="Id-ff25cb5a-936d-4ade-a446-e51c0f792738">
    <GetProductsByCategory xmlns="http://tempuri.org/fyj">
      <category>BIRDS</category>
     </GetProductsByCategory>
    </soap:Body>
  </soap:Envelope>
```

通过处理后的 SOAP 消息格式可以看出签名信息与加密信息已经成功添加到 SOAP 消息中。

5.3 特殊安全需求的实现

WSE3.0 的内置安全断言只能实现基本的消息交换安全。满足最常用的安全需求，并不满足所有的需求。对于特殊安全需求或其它需求，则必须通过自定义策略断言来实现。

目前，在本文中所考虑的特殊安全需求有以下四点：

（1）在用户为多角色情况下，客户端考虑使用多策略，而服务端则要实现相应的策略选择功能。

（2）在一个用户访问多项服务时，因服务业务不同，可能相应的安全需求也不同，则要考虑在一个策略下有根据服务方法不同而选择不同安全实现的功能。

（3）当需要对部分信息进行安全处理时，则需要实现能对部分 SOAP 消息进行安全操作的功能。

（4）为便于进行测试，需提供对 SOAP 消息的跟踪功能。

在设计自定义策略断言时，不仅要设计策略文件中的自定义策略断言，还要建立相应的自定义策略断言类，并通过灵活组合来实现所需的安全需求。

在设计自定义策略断言类时，根据同内置安全断言类结合的情况，可以有以下几种选择：

1．完全与内置安全断言无关

自定义策略断言中所有元素名及属性名都是自己定义，因此，对于该策略断言就必须完全自己进行解析，并处理所有自定义属性。适用于与消息交换安全无关的应用功能的实现。如辅助模块。

2．从内置安全断言中派生

自定义策略断言与自定义筛选器分别从内置安全断言和内置安全断言的内置筛选器中派生，并通过重载内置筛选器的方法来处理自定义的功能；同时，通过获取内置安全断言的默认策略扩展并进行对应修改，实现其余操作由基类自动完成。适用于在消息交换安全基础上附加特定需求的情况。

3．在自定义策略断言中包含内置安全断言元素

在解析自定义策略断言内部元素时，根据获取的内部策略扩展创建策略断言类实例；若为自定义元素和自定义属性，则自己进行解析和处理；若为内置断言元素，则会自动进行解析和处理。适用于实现复杂需求的情况。

本文在 CustomPolicyAssertionLibrary 程序集中实现了上述功能，对应策略集中的策略，提供了所需的自定义策略断言类。如图 5-4 所示：

图 5-4 自定义策略断言类集

（1） PolicyChoiceAssertion 类

在服务端接收请求消息和发送响应消息时，需要应用该类来处理消息。所以仅在 Web 服务端应用程序中存在。

（2） RequestBodyPolicy 类

在客户端输出请求消息及服务端接收请求消息时需要应用。所以不仅在

Web服务端应用程序中存在，还需要提供给客户端应用程序。

（3）PartsSignatureAssertion类

在服务端输出请求消息和客户端接收响应消息时，需要应用来处理消息。所以不仅在Web服务端应用程序中存在，还需要提供给客户端应用程序。

（4） CustomPolicyTraceAssertion类

在Web服务端和客户端都需要应用来实现对消息的跟踪。所以不仅在Web服务端应用程序中存在，还需要提供给客户端应用程序。

下面进行具体叙述。

5.3.1 多策略选择模块的分析与实现

1. 模块分析

对于一项Web服务，通常有不同用户来引用其服务，用户可能采用匿名或用户名/口令等多种方式来进行访问，造成在调用Web服务方法时所需要设置的安全需求不同。因此，需要针对不同用户为客户端设计多个安全策略供其选择引用，但是Web服务端一般一个服务提供类只能采用一个策略，为了能对客户端的不同策略都能进行响应，则在服务端策略中就需要设计能进行多策略选择的自定义断言来处理这种需求。显然，这时需要满足多种安全需求，属于复杂的需求的情况。可采用自定义断言元素中又嵌套断言元素的方式实现。

2. 实现步骤

（1）对应客户端所需的所有策略，对Web服务端的策略文件进行设置。如下：policyChoice自定义断言的简化声明信息，表明该断言中包含了3个策略断言可供选择，其中每一个断言对应客户端的一种策略。

```
<policyChoice>
<usernameForCertificateSecurity > … </usernameForCertificateSecurity>
<anonymousForCertificateSecurity > … </anonymousForCertificateSecurity>
<CustomSecurityAssertion > …  </CustomSecurityAssertion >
</policyChoice>
```

（2）对于该自定义策略断言类，重载ReadXml方法，对自定义策略断言元素内部进行解析。根据各个策略断言子元素名创建断言类实例，各自再通过重载ReadXml方法进行各自解析，并将策略断言放在策略断言集中。

流程图如图5-5所示：

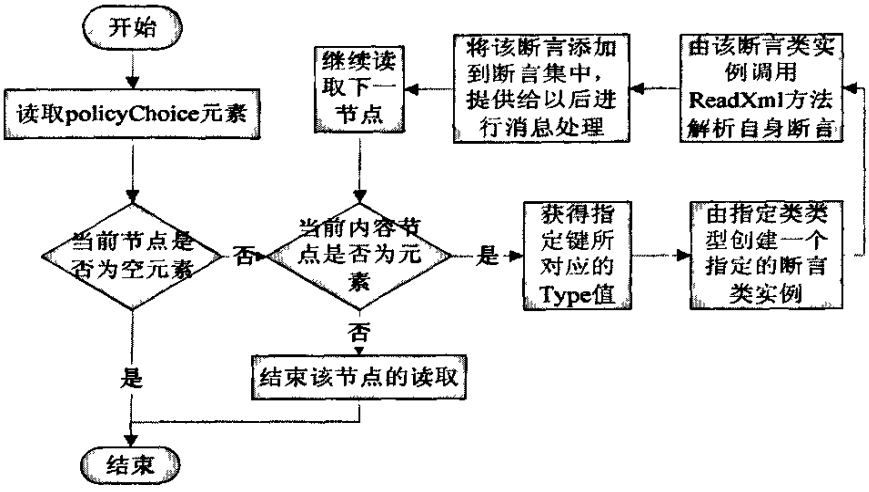


图5-5 自定义策略断言解析流程图

代码如下：

```
public override void ReadXml(XmlReader reader, IDictionary<string, Type> extensions)
{ reader.ReadStartElement("policyChoice");
  if (!reader.IsEmptyElement)
  {while (reader.MoveToContent() == XmlNodeType.Element)
    { extensions.TryGetValue(reader.Name, out PolicyExtension);
      object o = Activator.CreateInstance(PolicyExtension);
      assertion = (PolicyAssertion)o;
      assertion.ReadXml(reader,extensions);
      assertions.Add(assertion);
    }
    reader.ReadEndElement();
  }
}
```

（3）创建Web服务端的2个筛选器类。通过分别重载ValidateMessageSecurity方法和SecureMessage方法，对SOAP消息进行具体操作。

ValidateMessageSecurity方法实现用断言集中每个断言逐个创建筛选器来匹配处理接收消息的需求，若成功，则表示该断言就是对应目前客户端所用的策略断言，则保存该断言并退出。否则表示该断言不是要匹配的断言，

继续用下一个断言来匹配处理直至成功。

SecureMessage 方法实现用接收时匹配成功的断言创建的输出筛选器来处理输出消息。

5.3.2 不同方法不同需求处理模块的分析与实现

1. 模块分析

用户一般通过调用不同的Web服务方法来实现不同的业务功能。而每次Web 服务方法被调用时，就会产生一个消息。由于业务功能不同，所针对的数据信息也不同，也就有不同的消息保护需求，比如有的方法需要对消息体加密，而有些又不需要。因此，需要针对不同Web服务方法实现不同需求的处理。

通常，在内置安全断言中，为消息所提供的保护需求是被设置在内置安全断言元素的<protection>子元素中。WSE 3.0 允许通过检验消息体的第一个子元素来对一个Web服务方法映射保护需求。如下来自客户端的一部分请求消息：

```
<soap:Body>
  <GetCategories xmlns="http://tempuri.org/fyj" />
```

因此，可以基于消息体的第一个子元素的限定名来应用消息保护需求。显然，该功能是在基本安全需求上附加特定需求，属于上述选择中的第二种情况，可通过从内置安全断言派生自定义策略断言来实现。

2. 实现步骤

（1）对策略文件的设置

在 protection 内置元素中，在 requestAction 内置属性中设置相应的Web服务，同时根据不同要求设置对应的消息保护需求。

如下，针对 GetCategories 方法设置了在请求消息中加密 SOAP 体。则可针对其它方法的实际需要设置相应的保护需求（如不加密 SOAP 体等）。

```
<protection requestAction="http://tempuri.org/fyj:GetCategories">
  <request signatureOptions="IncludeAddressing, IncludeTimestamp, IncludeSoapBody"
encryptBody="true" />
  ...
</protection>
```

（2）创建从内置断言派生的自定义断言类，重载GetExtensions方法，获取内置安全断言的默认策略扩展，修改后形成该自定义断言的策略扩展集并注册。

通过设置自定义断言元素与内置安全断言元素相同的名称来实现修改，由于该断言的键所对应的type值已经发生改变，不是由内置安全断言类来处理，而是由派生的自定义断言类来处理，所以需要重载获得新的策略扩展来替代默认设置。如下：

```
public override IEnumerable<KeyValuePair<string, Type>> GetExtensions()
{
 Collection<KeyValuePair<string, Type>> result = new Collection<KeyValuePair<string,
 Type>>();
 //由基类执行 GetExtensions 方法来获取内置断言的默认策略扩展
 foreach (KeyValuePair<string, Type> extension in base.GetExtensions())
 {// 查找默认策略扩展中的指定键。usernameForCertificateSecurity 是
 UsernameForCertificateAssertion 内置断言的指定键名
   if ("usernameForCertificateSecurity".Equals(extension.Key))
   {//从本策略文件提供的 extensions 中获取新的 Type 值，形成新的策略扩展并替代同键名的
   原内置断言的策略扩展，并保留其它策略扩展项
   result.Add(new KeyValuePair<string, Type>("usernameForCertificateSecurity",
   this.GetType()));
   }
   else
   { result.Add(extension); }
 }
 //返回修改后的策略扩展作为自定义策略断言的策略扩展集进行注册
 return result;
}
```

（3）在该自定义断言所创建的输入筛选器中，调用该断言类中的GetOperationProtectionRequirements静态方法，从消息体中获取第一个子元素（即调用Web服务方法名），由该子元素属性信息形成protection区中属性的键形式，用来匹配配置文件中protection区内的requestAction键，获得其指定的安全需求。若匹配不成功，表示未指定自定义需求，则采用原

断言中的默认需求。如下：

```
static OperationProtectionRequirements GetOperationProtectionRequirements(
EndpointProtectionRequirements endpointProtectionRequirements, SoapEnvelope request)
    {if (request == null)
        {throw new ArgumentNullException("request");}
      XmlElement firstBodyElement = request.Body.FirstChild as XmlElement;//即方法名
      if (firstBodyElement == null)
        {throw new InvalidOperationException("…");}
  string              requestAction          =          String.Format("{0}:{1}",
firstBodyElement.NamespaceURI,firstBodyElement.LocalName);
        OperationProtectionRequirements result;
        //匹配配置文件中 protection 区内的 requestAction 键，获取需求项
         if (!endpointProtectionRequirements.Operations.TryGetValue (requestAction, out
result))
        {result = endpointProtectionRequirements.DefaultOperation;}
        return result;
    }
```

模块类结构图如图 5-6 所示；

图 5-6 类关系图

5.3.3 部分签名和部分加密模块的分析与实现

1. 模块分析

在很多实际情况下，并不需要对整个SOAP消息的正文进行签名和加密，而只是想对其中某一部分内容进行签名和加密，这可以通过向SOAP消息添加一个自定义 SOAP 头来实现。将要处理的内容从正文中分离出来，并在自定义SOAP头中定义，这样，对SOAP消息部分内容的签名和加密，就转变为对自定义SOAP头的签名和加密了。

2. 实现步骤

(1) WEB服务端的设置

在Web服务端，为了实现对自定义SOAP 标头的操作。首先需要创建一个从 SoapHeader类的派生类，并通过添加公共属性，来表示自定义SOAP 标头中的内容。要对自定义 SOAP 标头进行数字签名，这个标头必须要有一个 Id 属性，并且 Id 属性值必须是唯一标识符。这要通过应用 System.Xml.Serialization.XmlAttributeAttribute来指定，而其值则通过类的构造函数来获得，并保证其唯一性。

如下简要定义一个类：

```
public class  ProductPriceInfo : SoapHeader
{
 private string id;private string productid;private string productprice;
 public ProductPriceInfo(string ProductId) {
 this.ProductId = ProductId; id ="Id-"+ Guid.NewGuid().ToString();  }
 //设置 AttributeName 属性和 Namespace 属性
  [XmlAttribute("Id", Namespace = "…")]
 public string Id {  get { return id; }   set { id = value; }    }
 … }
```

(2) Web服务方法的设置

在Web服务代理类中声明一个自定义SOAP标头类的成员变量，使代理类中包含该标头类信息。并在 web 服务方法被调用前，设置该成员变量的SoapHeader 特性中的MemberName 属性，将该类成员信息形成所发送消息的标头，并设置为在请求和响应消息中都存在（默认只是请求时存在），使Web服务方法被调用时能响应处理该标头信息。

如下所示：

```
public ProductPriceInfo myHeaderMember;
[WebMethod]
[SoapHeader("myHeaderMember",Direction=SoapHeaderDirection.InOut)]
public Product GetProductInfoById(string Id) { … }
```

(3) 客户端方面的设置

在创建服务代理类对象后，再创建SOAP标头类对象，并将该SOAP标头对象加到服务代理类对象中的SOAP请求消息标头中，这样，在调用服务代理类的方法后，就会在发出的请求消息中附加该SOAP标头信息。

如下：`proxy.ProductPriceInfoValue = mySoapHeader;`

(4) 对自定义断言类的设置

实现对该自定义标头的签名的步骤如图5-7所示：

- 创建一个自定义策略断言类及对应的筛选器类。
- 在筛选器类的构造函数中，获取签名令牌和加密令牌。
- 在筛选器类中，重载SecureMessage方法。用于实现对自定义SOAP标头的数字签名和加密。
- 在该方法中，将签名令牌添加到SOAP头的security区中。
- 用该令牌创建一个MessageSignature类的新实例。
- 在消息头中根据元素名和命名空间值查找每一个元素，找到自定义标头元素后，获得其ID值，用它为自定义SOAP标头创建一个SignatureReference签名引用类的新实例，来接受带#Id前缀的Id字段的值。
- 使用AddReference方法将该实例加到MessageSignature元素中。
- 将修改好的MessageSignature元素加到SOAP头的security区中。
- 若这时需要对该自定义标头进行加密，则可以用加密令牌通过获得的ID值针对该自定义标头进行，然后将加密信息添加到消息头security区中。

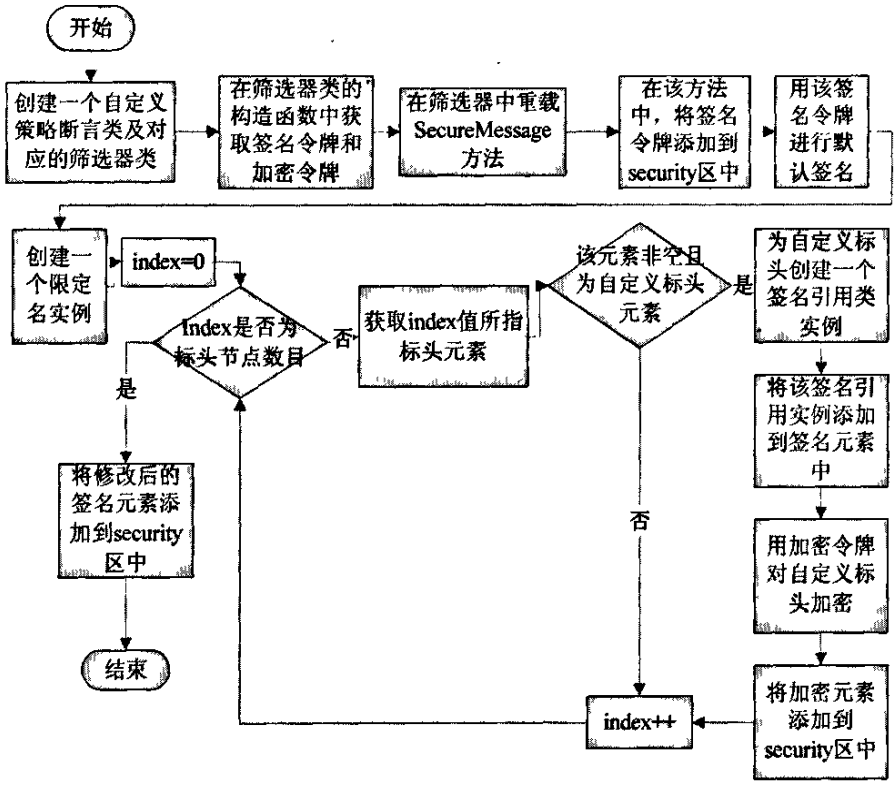


图 5-7 部分签名与部分加密消息处理流程图

该方法的简化代码如下：

```
public override void SecureMessage(SoapEnvelope envelope, Security security)
    {//在筛选器构造函数中获取 token 令牌用来签名，tokenb 令牌用来加密
        ...
        security.Tokens.Add(token);
        MessageSignature sig = new MessageSignature(token);
        //创建一个为 ProductPriceInfo 的限定名实例
        XmlQualifiedName header = new XmlQualifiedName("ProductPriceInfo",
        "http://tempuri.org/");
        for (int index = 0; index < envelope.Header.ChildNodes.Count; index++) {
      XmlElement child = envelope.Header.ChildNodes[index] as XmlElement;
          if (child != null && child.LocalName == header.Name && child.NamespaceURI ==
          header.Namespace)
          {
            //若找到该自定义标头，则通过创建一个签名引用来实现对其的签名
```

```
SignatureReference soapRef =new SignatureReference("#" +
child.Attributes[0].Value);
soapRef.AddTransform(new
Microsoft.Web.Services3.Security.Xml.XmlDsigExcC14NTransform());
sig.AddReference(soapRef);
//用 tokenb 令牌对该自定义标头进行加密
security.Elements.Add(new EncryptedData(tokenb, "#" +
child.Attributes[0].Value));}}
security.Elements.Add(sig); }
```

(5) 实现结果

在Web服务端处理业务后,形成的要发送的响应消息中,还未经签名和加密处理的自定义SOAP标头的简要格式如下所示:

```
<soap:Header>
<ProductPriceInfo p5:Id=" Id-e45ddc0a-1dfe-4329-bf4a-a07961cc17be " …>
<ProductId>1111</ProductId> …
</ProductPriceInfo>
</soap:Header>
```

经自定义策略类中的服务端输出 SOAP 筛选器进行部分签名和加密处理后，生成的消息头简要格式如下所示:

```
<soap:Header>
<ProductPriceInfo p5:Id="Id-e45ddc0a-1dfe-4329-bf4a-a07961cc17be" …>
<xenc:EncryptedData Id="Enc-9fa5f9e0-d6a9-469c-99ee-af202dbc1431" …>
</xenc:EncryptedData>
</ProductPriceInfo>
<wsse:Security soap:mustUnderstand="1">
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
<SignedInfo>
<Reference URI="#Id-e45ddc0a-1dfe-4329-bf4a-a07961cc17be">
<DigestValue>4JFIvAJ+AoRn5ZDhXYVZpjh0F2g=</DigestValue>
</Reference>
</SignedInfo>
</Signature>
```

```
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:ReferenceList>
<xenc:DataReference URI="#Enc-9fa5f9e0-d6a9-469c-99ee-af202dbc1431" />
  </xenc:ReferenceList>
 </xenc:EncryptedKey>
</wsse:Security>
</soap:Header>
```

通过两个消息头的比较，可以看到，自定义标头中的信息已经被成功加密，而针对它的签名信息也已经存在于 Security 区的 SignedInfo 元素中。

5.3.4 跟踪模块的分析与实现

1. 跟踪模块的分析

该功能属于辅助模块功能，因此设计时不必与内置安全断言有关，属于完全自定义的情况。通过使用自定义策略断言，将发送处理前后或接收处理前后 SOAP 消息保存在跟踪文件中，从而达到对在客户端和 Web 服务端之间交换的 SOAP 消息进行跟踪的目的。

在实现功能之前，需要在 Web 服务端和客户端进行相应的配置，两者是相互对应但又存在差别，同时还需要对各自的策略文件进行一定的配置，为自定义策略断言的实现做好准备。

2. 实现步骤

（1）在两端的配置文件中，为 Web 服务端和客户端注册 WSE 配置节处理程序，WSE 服务协议工厂和指定 Web 服务的策略文件。

（2）在两端的策略文件中，为 Web 服务端和客户端定义策略。

（3）对应策略声明，定义一个从 PolicyAssertion 类派生的自定义策略断言类。

（4）在自定义策略断言类中，根据跟踪需要，调用同一个创建自定义筛选器类的方法，在四个标准筛选器中选择进行创建。

如下，创建一个客户端输出筛选器：

```
public override SoapFilter CreateClientOutputFilter(FilterCreationContext context)
{ return new CustomTraceFilter(outputfile); }
```

注意：outputfile 为 XML 文件格式的文件名，需在类中指定。

如： string outputfile = "output.xml";

（5）在自定义策略断言类中，重载 ReadXml 和 GetExtensions 方法，解析策略文件中的自定义断言元素和属性，获取策略的配置信息（如跟踪文件对应的数据文件存储信息）。

（6）对应策略断言类的定义，从 SoapFilter 类中派生一个自定义筛选器类。并重载 ProcessMessage 方法实现将 SOAP 消息添加到跟踪文件的后面。

如：

```
public override SoapFilterResult ProcessMessage(SoapEnvelope envelope)
{//SoapEnvelope 代表了 SOAP 消息
    XmlDocument dom = null;
    DateTime timeStamp = DateTime.Now;
    XmlNode rootNode = null;
    dom = new XmlDocument();
    if (!File.Exists(filename))
    {//若跟踪文件不存在则创建一个新的 XML 文件作为跟踪文件
XmlDeclaration xmlDecl = dom.CreateXmlDeclaration("1.0", "utf-8", null);
dom.InsertBefore(xmlDecl, dom.DocumentElement);
rootNode = dom.CreateNode(XmlNodeType.Element, "log", String.Empty);
      dom.AppendChild(rootNode);
      dom.Save(filename);
    }
    else
    {//若跟踪文件存在则加载该文件
      dom.Load(filename);
      rootNode = dom.DocumentElement;
    }
    //将 SOAP 消息添加到跟踪文件中
    XmlNode newNode = dom.ImportNode(envelope.DocumentElement, true);
    rootNode.AppendChild(newNode);
    dom.Save(filename );
    //处理完毕，返回继续下一个筛选器的处理
    return SoapFilterResult.Continue;
```

}

该功能比较常用，也比较易于理解，属于XML文档的常用操作。因此省去流程分析。

5.4 实现方案的比较

应用WSE3.0的Web服务安全实现方案与传统Web服务安全实现方案的比较如下：

(1)传统方案中通常必须自己设计实现基本功能模块，并且须自己组合来实现安全消息交换。这样难免存在遗漏。而在本方案中可快速实现整合式安全消息交换。不仅节省了大量的重复设计时间，而且安全性更高。

(2)传统方案中安全代码与业务代码是混淆的，灵活性差。而在本方案中安全逻辑与业务逻辑是分离的，安全设计人员可专心进行安全功能的开发。

(3)传统方案中难于实现高级安全功能，可扩展性差。而在本方案中可以在内置安全功能的基础上方便地实现更高级的安全功能。

(4)本方案遵循了 WS-Security、WS-Trust 和 WS-SecureConversation 最新规范。互操作性更强。

5.5 本章小结

本章通过在一个实际系统中应用Web服务安全解决方案，验证了该解决方案的安全性。描述了该方案所能实现的基本安全功能模块及步骤，并详细分析和实现了在此基础上的高级安全功能模块及流程。

结 论

随着 Web 服务技术在分布式系统中的广泛应用，安全问题日益突出。传统的安全保护技术方法已经不能胜任，而一些常用的安全通信机制（如 SSL，TLS，IPsec 等）也不能满足 Web 服务的安全通信，如何提供安全可信的 Web 服务已成为 Web 服务应用必须解决的关键问题。本文就是在认识到该重要性的基础上，对 Web 服务安全性问题展开了研究和探讨。

本文分析研究了当前各种 Web 服务安全技术，重点介绍了 WS-Security 规范，同时对微软推出的支持 WS-Security 的开发工具 WSE 的工作原理及开发模型进行了深入的研究探讨。讨论了基于 WS-Security 安全技术的 Web 服务安全体系结构，同时在这个安全体系结构基础上，结合典型的 Web 服务应用模式和基于 Web 服务的应用支撑环境的特殊安全需求，提出了一种基于策略架构的消息层 Web 服务安全解决方案，对方案中的各个模块和流程做了详细说明，并在.NET 环境下，使用 WSE 3.0 实现了该方案。该安全解决方案基于 WS-Security 和 WS-Trust 协议，既保证了 SOAP 消息端到端的安全性，又通过安全策略文件提高了 Web 服务的互操作性，可满足 Web 服务应用环境下的安全需求。

目前已达到以下目标：

1. 该方案基于 WS-Security 规范并充分利用了 WSE3.0 的可扩展策略机制，具有一定的扩展性和灵活性。

2. 分析了基于策略架构的消息级 Web 服务安全性解决方案，并给出了安全消息交换的详细流程。

3. 分析了 WSE 安全性命名空间，通过部署策略文件与代码结合不仅实现了对 SOAP 消息进行身份认证、数字签名、加/解密等基本安全功能，还实现了一部分高级安全功能。

4. 利用多层设计模式将安全逻辑与业务逻辑相分离，使设计人员能专注于安全设计。

在不可信环境下实现 Web 服务安全是一个复杂的工作，由于水平和时间有限，本文对基于 WSE 3.0 的 Web 服务安全系统的研究和开发只做了部份工作。相信该模型有很多设计的不足之处，还需要进一步的完善。

在本文的研究过程中，作者认为该安全系统还有以下几点不足：

1. 无法在不可信环境下应用。

2. 该安全系统内部组件的开发基于.NET 框架，服务器端与客户端的运行必须依赖于.NET 框架。目前.NET 框架只能运行在 Windows 平台，真正的跨平台安装和运用有待于.NET 的通用性；

3. 目前系统的安全主要依赖于 WSE3.0 所提供的安全性解决方案。提供一个独立的安全解决方案以及分级的安全策略是下一步应该做的工作。

基于 Web 服务技术的各种应用还在不断的向前发展，Web 服务所面临的安全问题日益复杂和多样化，安全技术也在不断发展，本文提出的 Web 服务安全性机制也要相应地进行改进和扩展。具体地，进一步的工作将主要集中在以下几个方面：

1. 跟踪相关规范的发展，特别是微软和 IBM 提出的一系列 Web 服务安全性规范。这套规范尚在完善当中，需要密切关注，确保对规范新内容的正确理解。

2. 进一步研究更为复杂的 Web 服务应用模式(如 Web 服务工作流)，分析其安全需求，在现有工作的基础上设计和实现相应的解决方案。

3. 研究对 SOAP 消息交换的效率优化（如部分加密及压缩技术）。SOAP 是基于 XML 编码的，而 XML 文件其实就是一个文本文件，基于文本的数据比基于二进制编码的数据要远远大得多。因此对 SOAP 消息进行部分加密及压缩将会大大提高网络传输的效率。

4. 基于.NET 平台 SOAP 消息与 J2EE 平台消息的安全传递问题。本文研究的重点是基于.NET 平台，在.NET 平台上加密和签名的 SOAP 消息如何与 J2EE 平台进行交互是今后要仔细研究的课题。

总之，基于 WSE3.0 的 Web 服务安全系统是一个复杂的系统，它需要多种理论和技术的支持。需要做更多、更全面的研究和开发工作来完善和发展它。

致 谢

在硕士研究生学习生涯即将结束之际，首先我要深深地感谢我的导师唐慧佳副教授，在我读研的这几年中对我的不断支持和指导。为我创造了一个良好的学习环境。本人在硕士研究生学习期间以及撰写论文的过程中，尽管唐老师工作很忙，但从研究生课程的学习、论文选题、收集资料、总体构思、一直到最后定稿，唐老师自始至终都进行了悉心指导，并提出了很多宝贵意见，给予了我很大的关怀和帮助。是她的不断督促、严格要求，使得我得以较快的速度完成了论文写作，并通过写作本文，进一步掌握了科研工作的工作方法、工作思路。她渊博的学识、严谨的治学态度、诲人不倦的教学态度以及谦逊随和的人格魅力都让我受益匪浅，也影响着我的治学态度，学到了更多的为人处事的道理。在此，我对老师所付出的努力和给予的关心再次表示衷心的感谢和崇高的敬意！

感谢西南交通大学信息科学与技术学院的所有任课老师们。通过他们的授课使我能够较系统地学习了各门学科知识和学术研究的各种技能，提高了分析和解决实际问题的能力。

感谢我的室友和实验室的各位同窗，他们三年来在生活和学习上为我营造了一个安静、愉快和轻松的环境，给予我很多的关心和帮助。

我还要特别感谢我的父母和亲人，他们多年来对我学习和生活的各方面给予极大的支持、理解和鼓励，是我漫漫求学路上的力量源泉。他们对我的关怀和无私的奉献，鼓励着我克服困难，奋勇前进。我的每一份收获都包含他们的心血。

衷心感谢所有曾经关心和帮助过我的老师、同学和朋友，祝愿你们在以后的生活中身体健康，万事如意！

最后，谨向百忙中抽出宝贵时间评审本论文的专家、学者致以最诚挚的谢意！

参考文献

[1] 顾宁，刘家茂，柴晓路等著. Web Services 原理与研发实践. 北京：机械工业出版社,2006.1

[2] Peter Thorsteinson，G.Gnana Arun Ganesh[美]著. .NET 安全性与密码术. 梁志敏 蔡建译. 北京：清华大学出版社，2004.8

[3] 柴晓路，梁宇奇著. Web Services 技术、架构和应用. 北京：电子工业出版社，2003.6

[4] Bret Hartman[美] 等著. 全面掌握 Web 服务安全性. 杨硕 译. 北京：清华大学出版社，2004.6

[5] 微软公司 [美]著. ASP.NET 安全应用程序开发. 詹文军 等译. 北京：清华大学出版社，2003.8

[6] Matthew MacDonald 等著. C#数据安全手册. 崔伟 等译. 北京：清华大学出版社，2003.1

[7] Web Services Enhancements 3.0 for Microsoft .NET. URL:http://msdn.microsoft.com/webservices/webservices/building/wse/default.aspx. 2005. 12

[8] Patrick A.Lorenz[德]著. ASP.NET 2.0 大揭密. 王军、郭卫泳译. 北京：清华大学出版社，2004.5

[9] Keith Ballinger[美]著. .NET Web Services 架构与实现. 张晓坤译. 北京：电力出版社，2004.12

[10] David Jorgensen[美]著. 使用 XML 开发.NET Web 服务. 刘颖等译. 北京：科学出版社，2003.6

[11] Adam Sills[美]著. XML .NET 编程指南. 战晓苏译. 北京：电子工业出版社，2003.1

[12] Paul Kearney, Message level security for Web services, Information Security Technical Report, 2005(10): 41-50.

[13] UDDI org. Universal Description, Discovery and Integration v3.0.2. http://www.uddi.org/uddi-v3.0.2-20041019.pdf, 2004-10-19

[14] Nilo Mitra, SOAP Version 1.2 Part 0: Primer, W3C Working Draft

26 June 2002, http://www.w3.org/TR/soap 12-part0/

[15] W3C Standard. Web Services Description Language (WSDL) Version 1.2.
http://www.w3.org/TR/wsdl12/WSDL Version2_0 Core Language.htm.

[16] Blake Dournaee 著. XML 安全基础. 周永彬等译. 北京：清华大学出版社，2003.8

[17] Ben Galbraith, Whitney Hankison 等著. Web 服务安全性高级编程. 吴旭超，王黎译. 北京：清华大学出版社，2003.9

[18] Microsoft Corporation, Services(WSE) 2.0 SP3 for Microsoft .NET. http://msdn.microsoft.com/webservices/webservices/building/wse/default.aspx. 2005.2

[19]W3C Recommendation, XML Encryption Syntax and Processing. http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

[20] 陆昆仑等著. Web Service 编程-用 C#.NET 开发网络服务. 北京：希望电子出版社，2003.4

[21] 贾伟著. 网络与电子商务安全. 北京：国防工业出版社. 2006.4

[22]王景中，徐小青著. 计算机通信信息安全技术.北京：清华大学出版社. 2006.3

[23] 迁移到 WSE 3.0.
http://www.microsoft.com/china/MSDN/library/WebServices/WebServices/ServiceStation0509.mspx?mfr=true

[24] 企业应用的 Web 服务安全(WS-Security)技术之一.
http://www.matrix.org.cn/resource/article/2005-10-20/WS-Security_43880.html

[25] Web 服务和 Microsoft 平台.
http://www.microsoft.com/china/MSDN/library/WebServices/WebServices/wsmsplatform.mspx?mfr=true

[26] 增强 Web 服务安全性的新技术.
http://www.microsoft.com/china/msdn/library/security/mac0304WSSecu.mspx?mfr=true

[27] What Gives You the Right Combine the Powers of AzMan and WSE 3_0 to Protect Your Web Services.

http://msdn.microsoft.com/msdnmag/issues/05/11/default.aspx

[28] 使用 WSE 实现 Web Service 安全.
http://www.cnblogs.com/fineboy/archive/2006/03/23/356463.html

[29] WSE3.0 UsernameToken 应用.
http://www.cnblogs.com/RicCC/archive/2007/03/14/674736.html.

[30]WSE Security: Protect Your Web Services Through The Extensible Policy Framework In WSE 3.0.
http://msdn.microsoft.com/msdnmag/issues/06/02/WSE30/default.aspx.

[31] Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0.
http://go.microsoft.com/fwlink/?LinkId=57044.

[32] Web Services Enhancements(WSE)3.0 的新功能.
http://weblogs.asp.net/mfussell. 2005.6

[33] Protect Your Web Services Through The Extensible Policy Framework In WSE 3.0.
http://msdn.microsoft.com/msdnmag/issues/06/02/default.aspx.

[34]金丽娜,蒋兴浩,李建华.基于属性证书的Web Services访问控制模型.计算机工程. 2006, Vol.32 No.16

[35]黄政. 基于 ASP.NE 下 XML Web Services 安全机制的研究. 嘉兴学院学报. 2006, Vol.18 No.3

[36]孙延鹏 等. 基于 WS2Security 的 Web Service 模型的安全性实现. 计算机应用与软件. 2005, Vol.122 No.5

[37]祁涛等.Web 服务的安全性设计与实现.计算机与现代化.2006 年第 1 期

[38] 赵一鸣 等. Web Service 应用中的安全需求及实现身份认证的 3 种方法. 交通与计算机. 2005 年第 6 期 第 23 卷

[39]俞鹏. Web 服务安全问题的解决方案. 河南科技学院学报(自然科学版). 2005, Vol.25 No.3

[40]余国良, 韩文报. XML 的安全性研究. 信息工程大学学报. 2006. 8

[41]张维勇,程 俊,王建新.基于 WS-Security 安全规范的 Web 服务设计.合肥工业大学学报(自然科学版). 2006. 8

[42]杨艺清. Web Services 及 ASP. NET 应用程序的安全策略和方法. 湖南

科技学院学报. 2006, Vol.27 No.5

[43]孟伟，张璟，李军怀，刘海玲. Web 服务安全模型研究与实现. 计算机工程与应用. 2006.26

[44]窦永富，崔为红. Web 应用程序安全设计探析. 计算机系统应用. 2006.9

[45]丁达志，曹晓东，周勇. 基于.NET 的 Web Services 安全技术实现. 电脑开发与应用. 2006, Vol. 19 No. 2

[46]郭运宏，白春阳. 基于 Web Services 应用系统的安全性研究. 河南科技学院学报(自然科学版). 2006, Vol.34 No. 3

[47]李向，郭晓兰，严烨. 基于角色的 Web 系统安全策略研究. 计算机技术与发展. 2006, Vol.16 No.10

[48]金悦奇. 校园网 Web 服务安全建设的策略研究. 浙江师范大学学报(自然科学版). 2006, Vol.29 No.2

[49]李冬，郭荷清，韩涛. 一种面向 Web 服务的分层安全模型. 计算机工程与设计. 2006, Vol.27 NO.20

[50] 刘东伟，王成良. 一种面向商务系统的高安全性 W eb 服务综合安全模型. 计算机时代. 2006.2

[51]杨涛，刘锦德，谭浩. Web 服务安全基础设施的研究. 计算机应用. 2006, Vol.26 No.6

[52]韩涛，郭荷清. Web 服务安全模型的研究与实现. 计算机工程. 2006, Vol.32 No.10

[53] Signing the part of the message. http://66.129.67.100/247reference/__site/610

[54] Security handling based on Role of User and Destination of Service. http://www.topxml.com/Webservices%20Enhancements/rnl-17857-default.aspx

[55] Service Station Migrating to WSE 3_0. http://msdn.microsoft.com/msdnmag/issues/06/04/default.aspx

[56] Custom WSE 3.0 Policy Assertions for Signing and Encrypting SOAP Messages with Username Tokens. http://www.manbu.net/

[57] Securing .NET Web Services with the WS-Security Protocol. http://kb.csdn.net/

[58] XML-Signature Syntax and Processing .

http://www.w3.org/TR/2000/CR-xmld-core-20001031/

[59] XML Files Advanced Type Mappings.

http://msdn.microsoft.com/msdnmag/issues/03/06/default.aspx

[60] X.509Token.

http://blog.csdn.net/sxqem/archive/2006/6/15/799907.aspx

[61] wse3.0 problem with customtokenmanager.

http://forums.microsoft.com/MSDN/ShowForum.aspx

读硕士学位期间所发表论文和参加的科研项目

[1]付永军、唐慧佳. 基于 WSE 3.0 实现 Web 服务安全的应用. 成都信息工程学院学报. 2007.5（已录用）

[2]胡晓红、付永军. 基于策略的 web 服务安全解决方案研究. 微计算机信息. 2008.5（已录用）