PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

SEGURANÇA PARA WEB SERVICES COM CRIPTOGRAFIA HETEROGÊNEA BASEADA EM PROXY

SAMUEL CAMARGO DE SOUZA

Dissertação de Mestrado apresentada como requisito para obtenção do título de Mestre em Ciência da Computação pelo Programa de Pós-graduação da Faculdade de Informática. Área de concentração: Ciência da Computação.

Orientador: Avelino Francisco Zorzo

Porto Alegre, Brasil
2010

Dados Internacionais de Catalogação na Publicação (CIP)

S729s Souza, Samuel Camargo de
Segurança para web services com criptografia heterogênea baseada em Proxy / Samuel Camargo de Souza. – Porto Alegre, 2010.
87 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Avelino Francisco Zorzo.

1. Informática. 2. Segurança de Redes de Computadores. 3. Segurança de Dados. 4. Web Sites – Medidas de Segurança. 5. Servidor Proxy. I. Zorzo, Avelino Francisco. II. Título.

CDD 005.8

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "intitulada "**Segurança para WEB *Services* com Criptografia Heterogênea baseada em Proxy**", apresentada por Samuel Camargo de Souza, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 12/03/10 pela Comissão Examinadora:

Prof. Dr. Avelino Francisco Zorzo – Orientador — PPGCC/PUCRS

Prof. Dr. Duncan Dubugras Alcoba Ruiz – — PPGCC/PUCRS

Prof. Dr. Itana Gimenes – — UEM

Homologada em 15/06/2010, conforme Ata No. 010 pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

“A vida só pode ser compreendida olhando-se para trás;
mas só pode ser vivida olhando-se para a frente.”
Soren Keirkegaard

AGRADECIMENTOS

Gostaria de agradecer inicialmente ao meu orientador, Prof. Dr. Avelino Zorzo, pelo acompanhamento, motivação e amizade durante todo o período de mestrado.

A PUCRS, pela oportunidade e pela ótima infra-estrutura oferecida.

A HP, pelo ótimo ambiente de trabalho oferecido durante meu período de estágio e por todo o conhecimento adquirido durante esta temporada. Agradeço também a empresa parceira pelo financiamento desta pesquisa, que permitiu o seu andamento e conclusão.

A minha família pelo apoio incondicional, principalmente a minha mãe, principal motivadora e torcedora. Ao meu pai e meu avô, que não estão presentes na conclusão de mais este passo, mas sem os quais não teria ido tão longe.

A minha namorada, pelo apoio diário e pela motivação nas horas em mais precisei, muito obrigado também pela rigorosa revisão deste volume.

Enfim, a todos colegas, amigos que de perto ou de longe, de certo modo contribuíram para o bom andamento desta pesquisa.

SEGURANÇA PARA WEB SERVICES COM CRIPTOGRAFIA HETEROGÊNEA BASEADA EM PROXY

RESUMO

Atualmente, diversos serviços são disponibilizados pela Internet e há uma crescente demanda por estas aplicações. Considerando que a Internet não é uma rede segura, a confidencialidade de informações que circulam na rede é um fator de importância cada vez maior. Visando solucionar este problema, várias técnicas de segurança surgiram e cada empresa as adotou em suas políticas de uma forma diferente. Porém com a tendência de adoção do modelo de Arquiteturas Orientadas a Serviços, diferentes serviços com diferentes políticas de segurança necessitaram ser agrupados para que pudessem trabalhar em conjunto, o que acabou dificultando a criação de uma aplicação que segue um determinado padrão. Uma nova abordagem se faz necessária para facilitar a coexistência destas técnicas de segurança de uma forma produtiva. Assim, este trabalho apresenta um modelo voltado ao tratamento automático de características de segurança para serviços ordenados em Arquiteturas Orientadas a Serviço. Este modelo atua como um sistema intermediário capaz de "traduzir" as técnicas de segurança para um formato determinado. Com base neste modelo é apresentado um protótipo capaz de tratar automaticamente diferentes algoritmos de criptografia assimétrica, entre eles RSA, DSA e ECDSA. O objetivo do trabalho é testar a efetividade do modelo por meio de um estudo de caso, apresentando uma situação que este pode ser adotado por aplicações reais.

Palavras-chave: Web Service, Segurança, Proxy, SOA.

SECURITY FOR WEB SERVICES WITH HETEROGENEOUS CRIPTOGRAPHY PROXY-BASED

ABSTRACT

Currently, many different services are available through the Internet and there is a growing demand for these applications. Considering that the Internet is not secure, the confidentiality of such information is a factor of increasing importance. To overcome this problem, many security techniques have emerged and each company has adopted the policies in a different way, but with a tendency to adopt the model of Service-Oriented Architectures. Sometimes different services with different security policies need to be grouped to work together, which makes it difficult to create an application that follows this pattern. A new approach is needed to facilitate the coexistence of these techniques in a productive manner. Based on these requirements, this paper presents a model aimed at the automatic processing of security features to services ordered in Service Oriented Architectures, this model serves as an intermediate system able to "translate" the safety techniques for a particular format. Based on this model, it is presented a prototype able to automatically handle different asymmetric encryption algorithms, including RSA, DSA and ECDSA. This study is to demonstrate the effectiveness of the model through a case study, presenting a situation that can be adopted by real applications.

Keywords: *Web Service, Security, Proxy, SOA*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ACL	*Access Control List*
AES	*Advanced Encryption Standard*
API	*Application Program Interface*
B2B	*Business-to-Business*
BEEP	*Block Extensible Exchange Protocol*
BPEL	*Business Process Execution Language*
CORBA	*Common Object Request Broker Architecture*
DCOM	*Distributed Component Object Model*
DES	*Data Encryption Standard*
DSA	*Digital Signature Algorithm*
ECDSA	*Elliptc Curve Digital Signature Algorithm*
EDI	*Electronic Data Interchange*
HTML	*HyperText Markup Language*
HTTPS	*HyperText Transfer Protocol Secure*
IDL	*Interface Description Language*
JPEG	*Joint Photographic Experts Group*
NIST	*National Institute of Standards and Technology*
QoS	*Quality of Service*
RBAC	*Role-based Access Control*
REST	*Representational State Transfer*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Call*
RSA	*Rivest, Shamir e Adleman*
SAML	*Security Assertion Markup Language*
SMTP	*Simple Mail Transfer Protocol*
SOA	*Service-Oriented Architecture*

SOAP	*Simple Object Access Protocol*
SSL	*Secure Sockets Layer*
TCP	*Transmission Control Protocol*
UDDI	*Universal Description Discovery and Integration*
UTF-8	*8-bit Unicode Transformation Format*
W3C	*World Wide Web Consortium*
WSDL	*Web Service Definition Language*
WSS	*Web Services Security*
XML	*eXtensible Markup Language*

SUMÁRIO

1. INTRODUÇÃO

Sistemas computacionais estão se tornando cada vez mais indispensáveis à sociedade. Há várias décadas estes sistemas são responsáveis por atividades importantes como sistemas bancários e de gerenciamento de empresas. Com a expansão do uso da Internet, permitir que muitas outras aplicações como estas estejam acessíveis de qualquer lugar é uma necessidade e, portanto uma tendência.

O mercado de software tem realmente seguido a tendência, disponiblizando uma grande variedade de aplicações pela Internet. Entre as várias tecnologias existentes para a composição deste tipo de aplicação, Web Services [HBN+04] tem se consolidado como um padrão utilizado por um grande número de aplicações disponíveis na rede. Porém, a migração de aplicações para a Internet traz alguns problemas, entre eles está a segurança. Para utilizar determinados serviços é necessário efetuar cadastros, pagar com cartões de crédito e estas informações não podem ser de domínio público. Para contornar este problema foram criadas várias técnicas de segurança, como diferentes formas de autenticação e de criptografia.

As aplicações disponíveis através da Internet podem ser combinadas de diferentes formas, permitindo seu trabalho em conjunto e apresentando o conceito de Arquiteturas Orientadas a Serviço [MLM+06] que é utilizado em Web Services. Por exemplo, um sistema de pagamento em um *Web Site* de compras se comunica com o sistema de uma operadora de cartões de crédito, ambos os lados da operação devem codificar os dados utilizando o mesmo algoritmo de criptografia. Caso o algoritmo não seja o mesmo, não é possível que os dados sejam decodificados e devidamente compreendidos, impossibilitando desta forma a comunicação entre estes serviços.

Com base nas incompatibilidades entre algoritmos de criptografia para Web Services, este trabalho tem por objetivo propor um modelo que permita a integração de serviços, mesmo que possuam diferentes características de criptografia, de modo a tornar possível a comunicação entre eles de forma transparente, sem a necessidade de alteração. Para a realização desta tarefa, foi conduzido um estudo de várias tecnologias relacionadas com segurança para Web Services, bem como um levantamento do estado-da-arte nesta área de pesquisa.

Na análise dos trabalhos relacionados, foram levantadas muitas informações relevantes

para a definição e andamento desta pesquisa. As principais necessidades e características encontradas com relação a segurança de Web Services foram: interoperabilidade entre padrões já existentes; desacoplamento dos requisitos de segurança tanto do cliente quanto do servidor; um middleware auto-customizável foi a solução encontrada em diferentes casos para a interoperabilidade e também poderia ser aplicado a esta situação; e a falta de segurança na composição de serviços como um todo.

O estudo realizado indicou principalmente, que a flexibilidade do modelo Web Services, uma das suas principais vantagens, é uma das suas desvantagens também, isto se aplica ao modelo como um todo, porém no campo da segurança ela permite que uma variedade muito grande de técnicas sejam utilizadas de uma forma muito variada. Isto acaba por dificultar o processo de implementação de aplicações compostas por vários serviços.

Visando manter a flexibilidade proposta pelo modelo Web Services, permitindo que cada serviço defina as políticas de segurança da forma que desejar, foi proposto um modelo denominado *Proxy-Based Security* (PBSec), que prevê a conversão das políticas de segurança para apenas um formato, ele efetua esta conversão através de um serviço intermediário que é executado entre o cliente e o servidor.

A partir do PBSec, foi implementado um protótipo que prevê o tratamento automático de algoritmos de criptografia, permitindo que mesmo que a aplicação cliente não preveja o uso de determinado algoritmo o uso do serviço desejado seja possível. Para testar a efetividade do modelo e do protótipo foi efetuado um estudo de caso, em que uma aplicação orientada a serviço foi desenvolvida utilizando serviços com diferentes políticas de segurança.

Considerando as etapas de pesquisa realizadas, este trabalho está dividido em 5 capítulos. No Capítulo 2 é feita uma conceituação das tecnologias que serão abordadas. O Capítulo 3 apresenta o modelo e o protótipo do sistema. No Capítulo 4 é realizado um Estudo de Caso, onde é demonstrada a efetividade do modelo. Finalmente no Capítulo 5, são feitas as conclusões e apresentados os trabalhos futuros.

2. REFERENCIAL TEÓRICO

Este capítulo apresenta uma descrição dos principais conceitos e trabalhos relacionados com a proposta desta dissertação. A Seção 2.1 descreve o conceito de Arquiteturas Orientadas a Serviço; na Seção 2.2 é descrita a tecnologia Web Services e as especificações das tecnologias que a compõem; a Seção 2.3 descreve as tecnologias de segurança envolvidas neste trabalho; a Seção 2.4 lista e comenta os trabalhos relacionados e, finalizando este capítulo, são feitas as considerações finais relacionando as tecnologias com a pesquisa a ser apresentada.

2.1 Arquitetura Orientada a Serviço

SOA (*Service-Oriented Architecture*) [Erl05] é um tipo de arquitetura de *software* que utiliza processos de negócio e distribui funções em forma de serviços. Esta arquitetura fornece um conjunto de princípios que regem os conceitos utilizados durante as fases de desenvolvimento de sistemas e integração, tornando o resultado um pacote de funcionalidades baseadas em serviços interoperáveis. SOA é outra forma de integração entre módulos de software, ao invés de definir uma API, é definida uma interface que permite estabelecer a comunicação entre serviços e clientes. As principais características de SOA são:

- **Serviço** - é uma função de um sistema computacional, disponibilizada para outro através de uma interface bem definida.
- **Sem Estado (*Stateless*)** - é a não-dependência de um serviço para com outro (exceção para serviços coordenados).
- **Descoberta** - é a capacidade de localizar e identificar os serviços através de um repositório central.
- **Coordenação** - é a capacidade de organizar os serviços em uma sequência de modo a realizar uma determinada atividade (e.g. processos de negócio).
- ***Binding*** - é a forma de conexão dinâmica entre o cliente e o serviço.

SOA prevê que os provedores de serviço registrem as suas informações em um repositório central, incluindo as suas características, dados da empresa e uma descrição da interface de acesso. Este repositório deve ser utilizado pelo cliente para determinar as características dos serviços que procura, podendo, a partir deste momento, contratá-lo para iniciar a sua utilização [Erl05]. Este processo é análogo a um catálogo de serviços.

O conceito de SOA está fortemente ligado com o de Web Services [HBN+04], pois o seu modelo é um ótimo exemplo de SOA, uma vez que possui as características desta arquitetura e é a tecnologia mais aplicada para este caso. Porém, a implementação de um sistema SOA pode ser realizada utilizando qualquer tecnologia padronizada para *Web*, como CORBA (*Common Object Request Broker Architecture*) [Spe08], RMI (*Remote Method Invocation*) [WRW96], DCOM (*Distributed Component Object Model*) [Mic09] e REST (*Representational State Transfer*) [RR07]. Uma boa comparação para esta situação é mostrar SOA como se fosse um algoritmo. Ambos não pressupõem o uso de nenhuma tecnologia ou linguagem de programação específica, permitindo a implementação da forma como o desenvolvedor desejar.

Em soluções orientadas ao serviço cada tarefa pode envolver um número qualquer de serviços, gerando inclusive interdependência. Por exemplo, em um sistema de compra de passagens, um Serviço A fornece os destinos, horários e disponibilidades, um serviço B efetua as transações de pagamento e o serviço C emite as reservas. Desta forma, B depende da conclusão do serviço A para autorizar o pagamento e C depende de A e B para emitir e confirmar a reserva.

O escopo de uma atividade SOA pode variar bastante, indo de uma atividade simples, onde dois serviços se comunicam de forma síncrona utilizando mensagens, até uma atividade mais complexa, onde muitos serviços colaboram assincronamente para completar uma mesma tarefa. A Figura 2.1 ilustra a composição de serviços complexos, onde existem vários serviços (S1, S2, S3, S4, S5 e S6) que estão registrados em um servidor, a partir do qual é possível pesquisá-los e compor várias aplicações diferentes (A1, A2, A3, A4 e A5). Como pode ser visto na Figura 2.1, algumas aplicações podem ser formadas por serviços que geram uma dependência entre elas (A1, A3, A4, A5), enquanto outras ficam independentes (A2).

2.2 Web Services

Web Services [HBN+04] será a tecnologia utilizada para implementar o modelo descrito no Capítulo 3. A escolha por Web Services foi feita em virtude de ser o meio mais difundido para uma implementação SOA. Esta seção e as próximas subseções descrevem a tecnologia Web Services e as suas especificações adjacentes.

A W3C (*World Wide Web Consortium*) define um Web Service como:

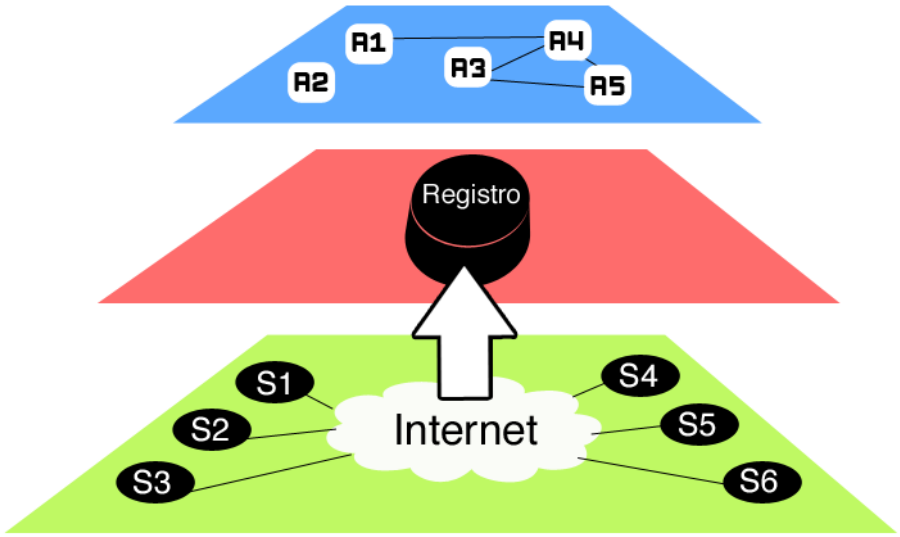

Figura 2.1: Composição de Serviços Complexos

> "Um sistema projetado para apoiar interações máquina/máquina através de uma rede. Deve possuir uma interface descrita em um formato compreensível pelas máquinas (especificamente WSDL (*Web Service Description Language*)). Outros sistemas interagem com Web Services de maneira pré-definida a partir da sua descrição, através de mensagens SOAP (*Simple Object Access Protocol*), tipicamente transportadas por HTTP com serialização XML em conjunto com outros padrões para Web [HBN+04]."

Basicamente, Web Services são funcionalidades de aplicações especificadas em XML (*Extensible Markup Language*) para outras aplicações, objetos ou bases de dados. Desta forma, ao utilizar XML, um programa envia uma solicitação a outro programa através de uma rede (normalmente a Internet) e recebe como resposta os dados solicitados.

Os padrões para Web Services definem os formatos para as mensagens e especificam como a mensagem será enviada. Estes padrões também descrevem as convenções de mapeamento de conteúdo das mensagens dentro e fora dos programas que implementam o serviço, também definem mecanismos para publicar e descobrir interfaces de Web Services. Web Services podem ser executados tanto em computadores *desktops* como em *handhelds*. Podem ainda ser utilizados para integração *business-to-business* (B2B), conectando aplicações que estão rodando em várias organizações em um mesmo aplicativo.

Os Web Services surgiram da necessidade de padronizar a comunicação entre plataformas distintas e linguagens de programação. Anteriormente, houveram tentativas de criar padrões como, por exemplo, o CORBA [Spe08], entretanto não tiveram êxito suficiente. Um problema nas comunicações foi o uso de RPCs (*Remote Procedure Call*) [Sri95] para realizar a comunicação entre os diferentes nodos. Além de apresentar certos problemas

de segurança, possuía a desvantagem da dificuldade de implementação em um ambiente como a Internet, principalmente por causa dos *firewalls*.

Em 1999, foi proposto um padrão que para permitir a comunicação entre diversas plataformas. Através de um consórcio entre empresas, foi desenvolvido o modelo Web Services. Após uma adoção significativa deste modelo, surgiu a iniciativa UDDI [CHvRR04]. O UDDI (*Universal Description Discovery and Integration*) é um diretório universal de descrição de serviços. Como o próprio nome sugere, tal iniciativa se propõe a listar todos os Web Services disponíveis na Internet. Nos anos seguintes, muitas linguagens de programação criaram APIs para o desenvolvimento de Web Service. Surgiram, então, padrões de desenvolvimento, QoS (*Quality of Service*) e controle de segurança, características responsáveis por tornar Web Services uma tecnologia cada vez mais sólida [New02].

A Figura 2.2 mostra como é formada a estrutura básica de um Web Service, ela é composta por: cliente, Web Service e diretório de registro de serviços, de forma que o cliente acessa o diretório de registro de serviços para escolher qual serviço deseja. Ao escolher o serviço, ele obtém um arquivo com a descrição da sua interface em uma linguagem apropriada e a partir desta descrição o serviço é solicitado. Após processar a informação desejada, o serviço retorna o resultado ao cliente.

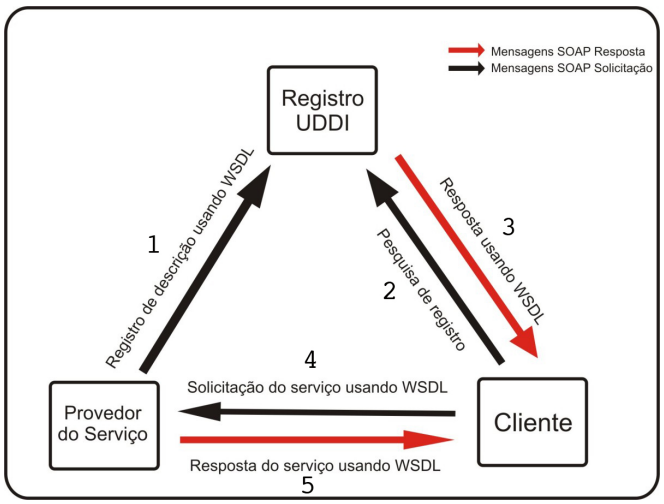

Figura 2.2: Arquitetura básica dos Web Services

A arquitetura básica de um Web Service utiliza os seguintes formatos/serviços:

- XML: formato de descrição das mensagens entre o cliente e o Web Services;
- SOAP: encapsula as mensagens XML em pacotes;
- HTTP: transporte dos pacotes SOAP;
- WSDL: linguagem responsável por descrever a interface do serviço;
- UDDI: responsável por armazenar a descrição do serviço.

2.2.1 Extensible Markup Language - XML

XML [PMSM+08] representa uma família de especificações relacionadas que são publicadas e mantidas pela W3C. Pode-se considerar o XML como a base sobre a qual os Web Services estão construídos, pois provê o armazenamento de descrições e determina o formato de transmissão para a troca de todos os dados via Web Services.

XML foi desenvolvida para superar as limitações do HTML (*HyperText Markup Language*), especialmente para melhorar o suporte à criação e gerenciamento de conteúdo dinâmico. O HTML é bom para definir e manter conteúdo dinâmico, mas não quando envolve uma plataforma de software na qual os dados associados necessitam ser gerados e compreendidos dinamicamente [PMSM+08]. Usando XML, pode-se definir qualquer número de eventos que associam significado aos dados, isto é, o dado é descrito e, em seguida, processado da forma desejada.

Esta grande flexibilidade implica em alguns problemas, pois permite a definição de esquemas pelo cliente, por isso é difícil assegurar que todos usarão os mesmos elementos da mesma maneira e com o mesmo significado. É exatamente nesta parte que é necessária a adequação a modelos.

Se dois computadores trocam dados XML, eles podem interpretar elementos da mesma forma se compartilharem as mesmas definições. Se dois nodos que compartilham um documento XML também compartilham o mesmo esquema, estes podem, certamente, dar o mesmo significado para os mesmos elementos da mesma forma.

O XML é uma família de tecnologias: uma linguagem de marcação de dados, vários modelos de conteúdo, modelos de *linking*, modelos de *namespaces* e vários mecanismos de transformação. Seguem abaixo os membros mais significantes desta família, para os Web Services [New02]:

- XML: É um formato para definição de elementos, atributos e *tags* dentro de um elemento no documento central, proporcionando um modelo de dados abstrato e um formato de serialização.
- *XML schema*: São documentos XML que definem os tipos de dados, conteúdo, estrutura e permitem elementos associados em um documento XML. Também é usado para descrever instruções de processos semânticos associados com documentos de elementos.
- *XML namespaces*: Sua função é unicamente qualificar nomes para elementos e aplicações de documentos XML.
- *XPointer, Xpath, XLink*: O *Xpointer* é um ponteiro para uma parte específica em um documento; O *XPath* são expressões para pesquisar em documentos XML; e o *XLink* serve para buscas múltiplas em documentos XML.

Uma das principais características da XML é possuir uma estrutura de documentos hierárquica. Um dos principais requisitos em Web Services é o reconhecimento da hierarquia XML e o mapeamento dos dados existentes dentro desta estrutura XML. A conversão de dados para XML é outro grande requisito da arquitetura XML. A Figura 2.3 ilustra a estrutura hierárquica de um documento XML. Através da interpretação dessa estrutura, os dados podem ser mapeados e transformados em outras estruturas diferentes. Este é o grande benefício para os Web Services, pois através deste recurso os dados de diferentes aplicações podem ser mapeados e agrupados dentro de uma aplicação maior, de forma a abrir portas para a arquitetura orientada a serviço (SOA).

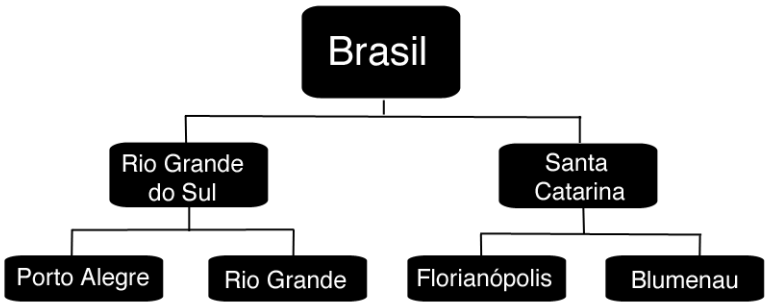

Figura 2.3: Estrutura Hierárquica de um documento XML

Atualmente, o XML é uma tecnologia amplamente adotada, principalmente em grandes sistemas como os ERPs, onde é necessária a adaptação da aplicação para cada empresa. Podemos considerar, então, que o XML pode ser utilizado em uma grande variedade de aplicações, como na formatação, serialização e transformação de dados. Web Services se comunicam trocando instâncias formatadas em documentos XML que contêm os dados e Web Services são descritos utilizando *XML Schemas* que definem os tipos de dados, estrutura e semântica envolvidos. Assim pode-se afirmar que o principal benefício proporcionado pelo XML aos Web Services é a independência das estruturas e dos tipos de dados, deixando livre a intercomunicação entre aplicativos [PMSM+08].

2.2.2 Web Service Definition Language - WSDL

WSDL foi originalmente criada pela IBM, Microsoft e Ariba, através do união de 3 propostas anteriores: *Microsoft's SOAP Contract Language*, *Service Description Language* e o *Network Accessible Service Specification Language*. Atualmente a WSDL está na versão 2.0 [CB07].

A WSDL estabelece um formato comum para descrever e publicar informações de serviços na Web. Utiliza elementos que contêm uma descrição de dados, que tipicamente utiliza

um ou mais esquemas XML, de modo a tornar as informações compreensíveis de ambos os lados da comunicação.

A especificação WSDL é frequentemente caracterizada por uma parte abstrata, conceitualmente análoga a IDL (*Interface Description Language*) convencional, e uma parte concreta, que define protocolos de conexão e outras informações, conforme a Figura 2.4 [ACKM03]. A parte abstrata é feita de definições do *port type*. Cada *port type* é uma coleção lógica de operações relacionadas. Cada operação define uma troca de mensagens simples. A mensagem é uma unidade de comunicação com um Web Service, representando os dados trocados em uma transmissão lógica simples. No caso de outros padrões Web Services, estas construções todas são definidas no XML.

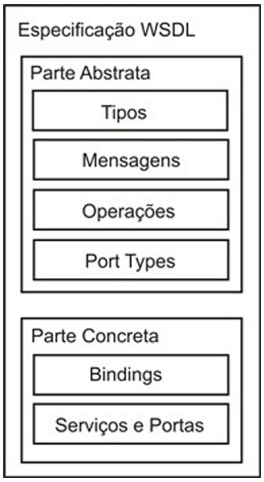

Figura 2.4: Especificação do serviço WSDL

Pode-se ainda acrescentar que WSDL é um padrão que pode ser utilizado com uma grande variedade de configurações. Por ser uma linguagem de descrição genérica, pode ser utilizada independentemente do sistema. Apesar de ser apontado como padrão, isto não significa que ele seja único. Podemos citar o EDI (*Electronic Data Interchange*) [oST93] como exemplo de outra linguagem de descrição, neste caso utilizada para descrição de atividades de manufatura.

2.2.3 Universal Description Discovery and Integration - UDDI

O UDDI (*Universal Description Discovery and Integration*) tem dois objetivos principais a respeito da descoberta de serviço. Primeiramente, apoia a pesquisa de informações por desenvolvedores, de forma que este saiba como escrever programas clientes que irão interagir com o serviço. Em segundo lugar, estabelece uma conexão dinâmica, permitindo que clientes solicitem o registro e obtenham as referências do serviço de interesse [CZC08].

Pode-se citar como objetivo do consórcio inicial a criação de um diretório mundial de descrição de serviços gratuito, onde todos poderiam publicar os seus serviços e pesquisá-los.

Com o tempo, esta iniciativa não teve muito sucesso e uma nova abordagem foi introduzida. Agora o UDDI caminha em direção a sua implementação em ambientes mais variados, de modo a permitir implementações locais (privadas) ou na Internet (normalmente públicas).

O UDDI proporciona uma maneira muito simples de pesquisa a seus dados. O diretório categoriza a informação contida em termos de acordo com o conteúdo do serviço, desta forma os dados podem ser localizados mais facilmente.

A analogia normalmente feita é com uma lista telefônica, onde as páginas brancas possuem uma lista de organizações, informações de contato e de serviços que estas organizações proporcionam. Nas páginas amarelas, há uma classificação dos serviços e das empresas de acordo com o tipo de serviço que oferecem. Já as páginas verdes descrevem como os serviços devem ser invocados. Esta organização normalmente é muito flexível e permite a fácil consulta de Web Services, bem como possibilita um fácil comparativo entre serviços similares oferecidos por uma mesma empresa [ACKM03].

A proposta do UDDI é que ele proporcione flexibilidade e extensibilidade para o registro e descoberta de serviços. Ele representa uma estrutura de dados que pode armazenar informações sobre diversas empresas e serviços, se instalado na Internet, ou apenas dos serviços privados, quando instalado dentro da rede interna de uma empresa. Portanto, o UDDI é a forma padronizada de se gerenciar metadados de Web Services.

2.2.4 Hypertext Transfer Protocol - HTTP

HTTP é um protocolo de transferência amplamente utilizado na Internet. Basicamente, ele especifica as mensagens que os clientes podem enviar aos servidores e quais as respostas que eles receberão. Por usar TCP (*Transmission Control Protocol*), o controle das mensagens (confirmações de entrega, reenvio de mensagens) é feito por este protocolo, deixando o HTTP livre para tratar melhor de outros assuntos. Ainda podemos acrescentar que a partir da versão 1.1, foi adicionada a função de conexões persistentes, que permite que mensagens adicionais sejam trocadas através de uma mesma conexão a TCP. Este tipo de conexão diminui o *overhead* das conexões [Tan02].

Apesar de ter sido projetado para a utilização na Internet, o HTTP foi criado visando futuras aplicações orientadas a objetos. Por este motivo ele aceita operações através de métodos. Foi por permitir este tipo de funcionalidade que o SOAP pôde surgir. Na Tabela 2.1 [Tan02] são descritos os métodos de solicitações HTTP.

Normalmente, estes métodos são seguidos por informações adicionais, chamados cabeçalhos. Estes cabeçalhos têm a função de proporcionar mais informações ao servidor ou ao cliente. Por exemplo, o cabeçalho *User-Agent*, informa sobre o navegador e a plataforma do cliente, já o cabeçalho *Content-Language*, informa o idioma utilizado na página Web. Através destes cabeçalhos é possível a gravação de *cookies* no cliente e adição de

Tabela 2.1: Métodos de Solicitações HTTP

Método	**Descrição**
GET	Solicita algum recurso
HEAD	Solicita a leitura de um cabeçalho
PUT	Envia certo recurso
POST	Envia dados para serem processados
DELETE	Remove determinado recurso
TRACE	Ecoa o pedido enviado
CONNECT	Serve para uso com um proxy que possa se tornar um túnel
OPTIONS	Consulta certas opções

alguns quesitos de segurança, como o provido pelo cabeçalho *Authorization*, que faz com que o cliente se identifique em páginas protegidas [Tan02].

Em Web Services, HTTP é o protocolo mais utilizado. Ele é o grande facilitador que permite que os Web Services se comuniquem de forma transparente através de *firewalls*. É também através dele que se acrescenta um protocolo de segurança muito importante, o HTTPS (*HyperText Transfer Protocol Secure*), que será abordado em mais detalhes mais à frente. Deve-se deixar claro que o uso do HTTP não é obrigatório quando falamos de Web Services, conforme pode ser visto em [SSV07], pode-se ainda utilizar SMTP (*Simple Mail Transfer Protocol*) [Pos82], BEEP (*Block Extensible Exchange Protocol*) [Ros01] e diversos outros.

HTTPS

O HTTPS (HyperText Transfer Protocol Secure) [RS99] é uma implementação do protocolo HTTP que adiciona uma camada de segurança, normalmente SSL, e permite que os dados sejam transmitidos em uma conexão criptografada. Ainda é possível verificar a autenticidade do servidor e do cliente através de certificados digitais.

Esta extensão do protocolo HTTP é utilizada normalmente quando os dados da conexão não podem ser vistos (e.g. compras, cartão de crédito, dados pessoais). A porta padrão para este protocolo é a 443.

2.2.5 Simple Object Access Protocol - SOAP

SOAP é uma das tecnologias mais importantes que fazem parte do padrão Web Services. Os Web Services não podem existir sem uma maneira abstrata de representar os dados e publicar as definições de interface. O SOAP tem uma das funções mais importantes: obter os dados de um local para outro sobre uma rede [ACKM03].

Este protocolo permite que o transmissor e o receptor de documentos XML tenham um

suporte comum para o protocolo de transferência de dados para que, desta forma, haja uma comunicação efetiva. Esta tecnologia é como se fosse uma pequena extensão do HTTP que suporta mensagens XML.

A especificação SOAP define como organizar informações utilizando XML de uma maneira estruturada e escrita que possa ser trocada entre pontos. Em [ACKM03] pode-se encontrar a seguinte definição:

> "Um grupo de regras que qualquer entidade que processe uma mensagem SOAP deve estar de acordo, definindo em particular os elementos XML que uma entidade deve ler e entender, bem como ações que estas entidades devem tomar se eles não entenderem o conteúdo."

Como um protocolo de comunicação, SOAP é desprovido de estado e mão única. Ele também ignora a semântica das mensagens que estão sendo trocadas através dele. A interação atual entre dois locais (*sites*) deve ser codificada dentro de um documento SOAP e qualquer padrão de comunicação, incluindo solicitação/resposta. Isto significa que o SOAP é criado para suportar aplicações fracamente acopladas que interagem através da troca de mensagens assíncronas em mão única com os outros. Qualquer complexidade adicional no padrão de comunicação, como mensagens síncronas em mão dupla ou interação estilo RPC requerem SOAP para serem combinadas com outros protocolos ou *middlewares* que possuem as propriedades adicionais. Por exemplo, para implementar uma chamada RPC convencional que toma parâmetros de entrada e retorna alguma saída usando SOAP, primeiro terá que codificar os parâmetros de entrada e a chamada para o procedimento dentro de uma mensagem SOAP. A resposta do procedimento deve também ser codificada dentro de outra mensagem SOAP. Finalmente, um protocolo de transporte síncrono como o HTTP deve ser utilizado para transportar as duas mensagens. Ele descreve como documentos devem ser escritos e organizados para que eles possam capturar a interação (e.g. RPC) e como mapear os documentos para os protocolos (e.g. HTTP) [STK01].

A troca de informações em SOAP acontece através de mensagens. Essas mensagens são utilizadas como um envelope onde a aplicação encapsula qualquer informação necessária para ser enviada. Cada envelope contêm duas partes, um "cabeçalho" e um "corpo". O cabeçalho é opcional, isto é, pode ou não estar presente em uma mensagem SOAP, e o corpo é obrigatório. Ambos, cabeçalho e corpo tem múltiplas sub-partes em forma de "blocos do cabeçalho" ou "blocos do corpo". Um bloco de cabeçalho (ou corpo) é qualquer filho de primeiro nível do elemento cabeçalho (ou corpo) de uma mensagem [GKM+07].

O último passo para tornar o SOAP funcional é definir como ele será transportado pela rede. Nenhum protocolo é imposto, no entanto ele é tipicamente associado ao HTTP. Para que esta comunicação funcione de forma efetiva é necessário determinar como a mensagem SOAP será empacotada e quais regras do protocolo de transporte são necessárias.

Para SOAP sobre HTTP, os modos mais comuns são o HTTP GET e o HTTP POST. A identificação do endereço de destino é necessária, para isto uma parte da mensagem SOAP é interpretada e incluída no HTTP, de forma que ele saberá o destinatário da mensagem. Um processo semelhante acontece quando é necessário o roteamento de mensagens SOAP. Os nodos pelo qual a mensagem deve ser roteada estão descritos em um *path* na mensagem SOAP e são interpretados para que seja possível o roteamento.

2.3 Segurança

Esta seção tem o objetivo de ligar os conceitos de segurança com os conceitos de Web Services apresentados na Seção 2.2. Serão abordadas algumas das tecnologias que estão atualmente em utilização para Web Services. São elas: Criptografia XML e *Web Services Security*. Inicialmente, esta seção descreverá rapidamente alguns conceitos sobre algoritmos de criptografia.

2.3.1 Algoritmos de Criptografia

Normalmente, algoritmos criptográficos são necessários para manter um sistema seguro, principalmente quando a comunicação acontece em uma rede não segura como a Internet. Há duas formas de criptografia mais comuns: as que utilizam algoritmos simétricos e as que utilizam algoritmos assimétricos. Ainda pode-se citar a funções *hash* como outra forma de criptografia.

Algoritmos Simétricos

Algoritmos de chave simétrica fazem referência a métodos de criptografia onde o transmissor e o receptor compartilham a mesma chave. Deve-se ressaltar que é possível que utilizem chaves diferentes, mas estas frequentemente estão relacionadas de uma maneira simples.

Este tipo de algoritmo é dividido em cifras de fluxo ou em cifras por bloco. As cifras de fluxo cifram os bits da mensagem individualmente, enquanto as cifras por bloco criptografam vários bits como se fossem um. A cifragem por blocos é mais frequente, normalmente são utilizados blocos de 64, 128 e 256 bits, sendo que este número pode variar indefinidamente. Os algoritmos simétricos são mais rápidos, devido à menor complexidade das operações necessárias para decriptografar os dados.

Podemos citar como exemplos de algoritmos simétricos: DES (*Data Encryption Standard*) [NIS99], Triple-DES [NIS99], AES (*Advanced Encryption Standard*) [NIS01], TwoFish [SLLCJW07] e BlowFish [Sch94].

Algoritmos Assimétricos

Algoritmos assimétricos, ou algoritmos de chave pública, utilizam um par de chaves: uma pública e outra privada. A chave pública é distribuída para os transmissores da mensagem (e.g. pessoa, sistema), já a chave privada é mantida apenas pelo receptor, sendo que apenas o proprietário desta chave é capaz de decriptografar a mensagem codificada a partir da chave pública.

É possível que este tipo de algoritmo seja utilizado de duas formas: para confidencialidade (conforme exemplo citado no parágrafo anterior), a chave pública é utilizada para cifrar a mensagem, sendo que apenas a chave privada pode decifrá-la; e para autenticidade, a chave privada é utilizada para criptografar a mensagem. Isto garante que apenas o dono desta chave poderia ter cifrado a mensagem que será decifrada com a chave pública.

Os principais algoritmos de criptografia assimétrica são RSA (*Rivest, Shamir e Adleman*) [RSA78], Diffie-Hellman [oEE00], DSA (*Digital Signature Algorithm*) [oST94] e a sua variação com curvas elípticas ECDSA (*Elliptic Curve Digital Signature Algorithm*) [oST94]. Apesar de haver vários algoritmos de criptografia assimétrica, os três seguintes foram escolhidos por serem suportados pelo BouncyCastle (API de criptografia que será introduzida juntamente com a explicação do protótipo na Seção 3.3):

- **DSA** (*Digital Signature Algorithm*) [oST94] é um padrão do governo dos Estados Unidos para assinaturas digitais. Foi proposto pelo NIST (*National Institute of Standards and Technology*) em agosto de 1991. A sua autoria é atribuída a David W. Kavitz. Este também é um algoritmo assimétrico que utiliza um par de chaves, porém a geração de suas chaves é feita através de funções *hash*.

- **RSA** [RSA78] é um algoritmo de chaves assimétricas que fundamenta-se em teorias clássicas dos números. Seu nome vem das iniciais dos nomes do professores que inventaram o algoritmo, Rivest, Shamir e Adleman. RSA utiliza um par de chaves, uma pública e outra privada. Todas as mensagem são criptografadas com a chave pública e só podem ser lidas com a chave privada. Este algoritmo é considerado uma das melhores implementações de criptografia assimétrica.

- **ECDSA** (*Elliptic Curve Digital Signature Algorithm*) [oST94] é uma variante do algoritmo DSA que utiliza criptografia através de curvas elípticas. Está contida na proposta do NIST que também abrange o DSA. A diferença entre os algoritmos está na quantidade de bits utilizados pela chave pública, o que proporciona o dobro de segurança em relação ao DSA.

2.3.2 Criptografia XML

A criptografia XML é uma das melhores formas de garantir a confidencialidade das informações durante o seu trânsito em uma rede. A grande vantagem da Criptografia XML, se comparada a outras formas como SSL (*Secure Sockets Layer*) [FKK96], é que ela permite que os dados sejam criptografados mesmo que a transação SOAP tenha que ser roteada durante o caminho, além de manter-se criptografada após ser processada em um Web Service [O'N03].

A W3C ressalta duas funcionalidades da criptografia XML. A primeira é que os dados criptografados podem ser expressos utilizando XML, e a segunda é que apenas partes do documento podem ser criptografadas. Para que esta criptografia seja feita, vários algoritmos bem conhecidos podem ser utilizados, como DES, Triple-DES e RSA.

Segundo [O'N03] os seguintes tipos de informações podem ser expressos utilizando Criptografia XML:

- Detalhes do tipo de dados contidos no documento criptografado (por exemplo, JPEG (*Joint Photographic Experts Group*), XML, HTML);
- Uma chave criptografada (por exemplo, um par de chaves assimétricas);
- Informações sobre o algoritmo da chave que foi escolhida (por exemplo, Diffie-Hellman);
- Referência aos dados criptografados;
- O método de criptografia utilizado.

O trecho de um arquivo XML, representado na Figura 2.5, possui informações sobre uma compra através de cartão de crédito e necessita que seus dados sejam criptografados para que ninguém tenha acesso a estas informações indevidamente.

Como podemos ver na Figura 2.6, o trecho que possuía informações sobre o cartão de crédito foi substituído por um campo *<EncriptedData>* que está criptografado. Dentro deste campo, temos o campo *<CipherValue>* que possui os dados cifrados, propriamente ditos. Desta forma, é possível transferir os dados de uma maneira segura. Caso alguém intercepte este arquivo, não terá acesso aos dados, pois para visualizá-los é necessária a chave criptográfica.

Vejamos quais são os passos para que a criptografia descrita no exemplo seja realmente executada:

1. Escolher o Algoritmo de Criptografia;
   Primeiro é necessário escolher o algoritmo de criptografia que será utilizado. As opções mais utilizadas representam algoritmos muito bem conhecidos, como os algoritmos de criptografia assimétrica.

```
<?xml version='1.0'?>
    <PaymentInfo xmlns='http://www.pucrs.br/loja'>
        <Name>Samuel Souza</Name>
        <CreditCard Limit='12.000' Currency='EUR'>
                <Number>1234 5678 9012 3456</Number>
                <Issuer>Local Bank</Issuer>
                <Expiration>12/08</Expiration>
        </CreditCard>
    </PaymentInfo>
```

Figura 2.5: Exemplo de arquivo XML que necessita de criptografia

```
<?xml version='1.0'?>
    <PaymentInfo xmlns='http://www.pucrs.br/loja'>
        <Name>Samuel Souza</Name>
        <EncriptedData Type='http://www.w3c.org/2001/04/xmlenc#Element'
        xmlns='http://ww.w3c.org/2001/04/xmlenc#'>
                <CipherData>
                        <CipherValue>A23B45C56</cipherValue>
                </CipherData>
        </EncriptedData>
    </PaymentInfo>
```

Figura 2.6: Exemplo de arquivo XML criptografado

2. Obter e Representar a Chave de Criptografia;
   Este passo é necessário para que o destinatário possa decriptografar o arquivo. Para tanto, é necessário o envio, através de um meio seguro, da chave de criptografia.

3. Serializar os Dados com a Codificação UTF-8 (*8-bit Unicode Transformation Format*) [Yer96];
   Para que se possa criptografar os dados, é necessário que eles estejam organizados em octetos, caso não estejam, deve ser aplicada a codificação UTF-8 para que eles fiquem organizados de acordo com a necessidade.

4. Executar a Criptografia;
   Tendo feito todos os passos anteriores, agora já é possível executar o processo de criptografia.

5. Especificar os Tipos de Dados;
   A especificação dos dados não é obrigatória, mas é muito recomendada para que o receptor saiba que dados ele está decriptografando.

Segue abaixo a descrição do processo de decodificação.

1. Determinar o Algoritmo, Parâmetros e as Informações da Chave;
   Este item não é obrigatório, pois normalmente o decodificador já tem conhecimento destas informações.

2. Localizar a Chave;
   Para localizar a chave, é necessário analisar o corpo da mensagem SOAP onde estão as informações sobre como aquela chave foi enviada.

3. Decodificar os Dados;
   Neste passo, é possível decodificar os dados contidos no campo *<CipherValue>*.

4. Processar os elementos XML;
   Este processo transforma os dados obtidos novamente em arquivos XML; caso os dados não sejam XML, este passo pode ser omitido.

5. Processar os dados que não são elementos XML;
   Finalmente, os dados então são repassados para a aplicação final, que pode interpretá-los.

Para encerrar a seção sobre Criptografia XML, ressalta-se que esta é uma tecnologia relativamente fácil de ser implementada, pois possui o apoio de diversas bibliotecas e softwares, como o *IBM XML Security Suite* e o *Microsoft Web Services Enhancements*. Apesar de ser de fácil implementação, a criptografia XML não é ideal para todas as situações, pois o *overhead* causado por este modelo pode ser considerado alto, o que pode inviabilizar o projeto [O'N03].

2.3.3 Web Service Security - WSS

O principal objetivo dos Web Services é a comunicação entre sistemas de diferentes empresas. Conforme mencionado anteriormente, esta é uma questão muito sensível, pois empresas utilizam diferentes plataformas, linguagens de programação e tecnologias de segurança. A diversidade de sistemas operacionais e linguagens de programação é aceita em Web Services através de XML. Desta forma, o padrão WSS estabelece alternativas de segurança mais complexas que as tradicionalmente utilizadas com Web Services, proporcionando mais segurança na comunicação em nível de mensagens [O'N03].

O WSS foi inicialmente proposto pela Microsoft em 2001, sendo que a sua principal característica é segurança em mensagens SOAP, o que difere este dos outros padrões de segurança apresentados anteriormente. Desde a sua proposição, outros padrões entraram nesta especificação, como o *WS-Trust* [NGG+07b] e o *WS-Federation* [KM06]. Estes padrões posteriores normalmente são chamados "WS-*" e serão abordados nesta seção. Desde 2002 é um padrão gerenciado pela OASIS [Sin07].

WSS define a forma como os *tokens* de segurança podem ser incluídos em mensagens SOAP e como especificações de segurança XML podem ser utilizadas para criptografar estes *tokens*.

Conforme pode ser visto na Figura 2.7 [O'N03], o WSS está situado sobre a camada SOAP. Para isto, as características providas (criptografia e autenticação) são proporcionadas utilizando Assinaturas XML e Criptografia XML (abordado na Seção 2.3.2). Para facilitar o entendimento, seguem breves explicações individuais das especificações da pilha WSS.

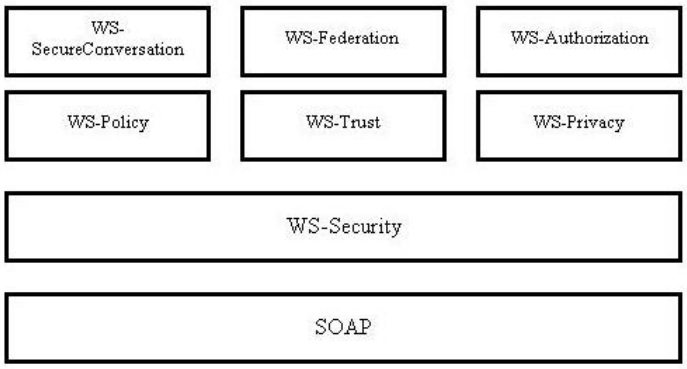

Figura 2.7: Especificações Web Services Security

WS-Policy

A especificação WS-Policy permite que organizações que estão publicando os seus Web Services especifiquem seus requisitos de segurança. Estes requisitos incluem algoritmos de criptografia, assinaturas digitais e a forma como estas informações devem ser integradas ao Web Service [HYO+07].

O WS-Policy permite que os requisitos de segurança sejam descritos. Função semelhante a WSDL que ao invés da segurança descreve o próprio Web Service. Desta forma, as empresas podem entender os pré-requisitos necessários para a utilização do seu Web Service.

WS-Trust

O WS-Trust define como relacionamentos confiáveis podem ser estabelecidos. Este relacionamento pode acontecer de duas formas: direto ou intermediado. No caso de relacionamento intermediado, um *proxy* é utilizado para avaliar as necessidades impostas pelo WS-Policy e solicitá-las às entidades que desejam utilizá-lo como ponto intermediário [O'N03]. Este *proxy* é considerado uma entidade confiável, que verifica os *tokens* de segurança, já o WSS é utilizado para transferir estes *tokens* utilizando assinatura e criptografia XML para garantir a confidencialidade. Este modelo permite delegação de responsabilidades, pois quando recebe uma mensagem SOAP, ele age como se fosse o usuário final, garantindo assim que os requisitos mínimos estejam especificados.

WS-SecureConversation

WS-SecureConversation [NGG+07a] permite que o solicitante e o Web Service se autentiquem utilizando mensagens SOAP e, desta forma, estabeleçam um contexto de segurança, onde estão mutuamente autenticados. Esta especificação funciona de forma similar ao SSL para o HTTP, porém, é utilizado com o protocolo SOAP.

WS-Privacy

O WS-Privacy utiliza-se da combinação de WS-Policy, WS-Security e do WS-Trust para comunicar quais são as políticas de privacidade. Estas políticas de privacidade são definidas pela empresa que desenvolve o Web Service.

Este padrão especifica como os requisitos podem ser incluídos nas descrições WS-Policy. Utiliza o WS-Trust para avaliar os dados encapsulados dentro de mensagens SOAP. Estas relações entre padrões significa que todas as especificações WSS têm dependência entre si [O'N03].

WS-Federation

A WS-Federation é uma especificação composta pelo WS-Security, WS-Policy, WS-Trust e pelo WS-SecureConversation. A função é criar uma federação que permita a interação entre partes que não possuem as mesmas regras de autenticação. Por exemplo, um domínio autentica apenas utilizando Kerberos e outro domínio apenas utilizando X.509; a federação permitiria que um usuário se autentique em um domínio e utilize o serviço em outro domínio [O'N03].

WS-Authorization

O WS-Authorization descreve como políticas de acesso para um Web Service são especificadas e gerenciadas. Para a realização deste serviço é utilizado um formato de autorização e uma linguagem de autorização, tais como: *ACL-Based Authorization* (*Access Control List*) e *RBAC-based Authorization* (*Role-based Access Control*) [O'N03].

2.4 Trabalhos Relacionados

Com o objetivo de entender e levantar o estado da arte no quesito segurança em SOA foram analisadas diversas iniciativas de pesquisa, direta e indiretamente relacionadas com este trabalho. Existem diversas pesquisas na área de *Web Services*, mas poucas são relacionadas com a interoperabilidade entre serviços com diferentes características de segurança:

- Em [AGL+07], os autores propõem a discussão de interoperabilidade entre diferentes arquiteturas de federações, que são mecanismos que permitem que diferentes domínios compartilhem a confiança na autenticação de identidade e atributos. É realizando um comparativo entre dois modelos (SAML (*Security Assertion Markup Language*) e *WS-Federation*) que permite definir uma forma de convergência entre eles. Esse artigo possui um objetivo semelhante ao que é utilizado por esta pesquisa, porém os alvos são diferentes: ele trata de interoperabilidade em federações com *tokens* de autenticação, enquanto tratamos de interoperabilidade de algoritmos de criptografia em SOA.

- Em [MMNS06] é proposto um *framework* orientado a aspectos para Web Services, onde seria possível tornar a implementação de características de segurança mais flexível. O proposto por esse artigo difere do proposto nesta pesquisa, pois trata de uma forma diferente de desenvolvimento de Web Services visando o reuso de código e não uma forma de interoperabilidade de especificações. Apresenta idéias interessantes, como a separação da aplicação em si de quesitos de negócio como segurança e autenticação. A relação do artigo com o trabalho que será apresentado acontece através da busca por uma separação similar, porém a aplicação será separada da segurança através da utilização de serviços diferentes e não de programação por aspectos.

- Em [HYS06], os autores utilizam a mesma técnica de [MMNS06], ao aplicar aspectos com o objetivo de separar o controle da segurança de outros requisitos funcionais da aplicação. Segundo o autor, esta separação torna a composição de Web Services mais flexível e dinâmica. Porém, a separação em aspectos visa o reuso e busca facilitar o desenvolvimento de aplicações com segurança através da separação das características de segurança.

- Em [CM05], são apresentadas deficiências nas especificações Web Services em composição de serviços. É citado que a linguagem BPEL (*Business Process Execution Language*) [JE07], que especifica interações entre Web Services, não suporta propriedades como: segurança, confiabilidade e persistência. Como solução para este problema é apresentado um *middleware* de integração transparente que expande a especificação BPEL, provendo estas propriedades ao protocolo. O conceito de um sistema intermediário, transparente ao usuário capaz de tratar dificuldades com políticas de segurança, é útil para a composição do modelo a ser apresentado.

- Em [EMT06], os autores apresentam um *middleware* que permite a auto-adaptação de políticas de composição de serviços permitindo, desta forma, que ele seja auto-customizável independentemente das políticas utilizadas. Este conceito é interessante, pois pode ser estendido a algoritmos de criptografia, de maneira que o sistema

permitiria que, apesar de diferentes algoritmos estarem sendo utilizados, o sistema seria capaz de se auto-configurar permitindo a comunicação entre eles.

- Utilizando os conceitos apresentados em [CM05] e [EMT06], em [JZ08] é apresentado um modelo de segurança baseado em *proxy* similar ao modelo que será apresentado no próximo capítulo. Em [JZ08], os autores propõem a implementação de um sistema intermediário capaz de tratar diferentes características de segurança, como formas de criptografia e autenticação. Porém, para que o sistema funcione corretamente, é necessário que o mesmo esteja instalado em ambos os lados da conexão (cliente e servidor). A forma de interação com os serviços é através da captura do fluxo de dados da placa de rede. Os resultados apresentados não descrevem testes com diferentes algoritmos de criptografia e os algoritmos utilizados não são mencionados o que dificulta uma comparação com os seus resultados. Porém, o modelo é muito similar com o que será proposto aqui, testando a efetividade do mesmo.

- Em [IR09], é descrito um método de segurança para *Web Services* que visa proteger aplicações Web Services em que o *front-end* é acessado em *web browsers*. Segundo o artigo, a maior parte das APIs SOAP oferecidas com os *browers* não são compatíveis entre si, dificultando uma solução uniforme para a segurança na comunicação. O artigo apresenta um método através do qual a comunicação entre o *browser* e o cliente é protegida fim-a-fim utilizando WSS. A utilização do WSS e de um sistema externo ao serviço principal para prover segurança são as principais contribuições do artigo para com o trabalho que será apresentado nos próximos capítulos.

- Em [FH08], são descritas diversas vulnerabilidades da arquitetura SOA, de modo que divide-os em níveis de segurança e ressalta seus riscos. Aborda soluções apenas no nível que considera menos estudado, o nível de negócio, e propõe novas abordagens de segurança através da linguagem BPEL. Isto significa que a segurança é mais crítica na composição de serviços, o que contribui com este trabalho uma vez que corrobora o nicho escolhido. Tal nicho é o tratamento de características de segurança em Web Services complexos, ou seja, compostos por vários serviços.

A Tabela 2.2 apresenta concisamente as contribuições feitas por cada um dos artigos analisados nesta seção. Os campos são marcados de acordo com a relevância da contribuição para com esta dissertação. Dentre todos os assuntos abordados foram selecionados os 4 tópicos de maior relevância para o andamento deste trabalho de pesquisa.

Tabela 2.2: Contribuições dos Trabalhos Relacionados.

Contribuições	[AGL+07]	[MMN06	[HYS06]	[CM05]	[EMT06]	[JZ08]	[IR09]	[FH08]
Interoperabilidade	X					X		
Desacoplamento da Segurança		X	X			X	X	
Middleware Auto-customizável				X	X	X		
Falta de Segurança em SOA				X				X

2.5 Considerações Finais

O objetivo deste capítulo foi realizar um estudo direcionado às tecnologias relacionadas com Web Services e suas formas de segurança, de modo a servir de referência para o modelo, protótipo e estudo de caso, apresentados nos Capítulos 3 e 4.

O primeiro passo do trabalho foi identificar a tecnologia Web Services, arquitetura de referência na qual está baseada (SOA), e tecnologias que o compõem, fazendo referência as características que puderam ser consideradas relevantes na identificação do problema a ser apresentado na Seção 3.1. Foram avaliadas as tecnologias relacionadas com Web Services e SOA, buscando diferenciar conceitos e tornar claro que SOA é uma arquitetura de referência e que Web Services não são sua única forma de implementação, pode-se ainda usar CORBA, RMI, entre outros.

O passo seguinte foi identificar formas de estabelecer segurança durante a comunicação de serviços. Considerando que a forma mais comum de segurança é a criptografia, foram levantadas as suas formas possíveis. Ao estudar a Criptografia XML, chegamos a um padrão que a contêm, porém muito mais complexo: o *Web Services Security* (WSS). O WSS é composto por vários outros padrões, que estabelecem convenções e especificam formas de uso para diferentes formas de segurança.

Durante o estudo destas tecnologias foram identificadas dificuldades de desenvolvimento de aplicações que utilizam estes padrões de segurança. Mesmo seguindo as especificações, o desenvolvedor deve fazer muitas escolhas, entre as quais está o variado número de algoritmos de criptografia, o que permite que as empresas criem políticas de segurança diferentes. A maior dificuldade, no entanto, está em utilizar aplicações com diferentes políticas de segurança em uma arquitetura SOA, onde múltiplos serviços são orquestrados para comporem diferentes aplicações.

A complexidade de diferentes políticas de segurança acaba por tornar difícil o gerenciamento de vários certificados de segurança, principalmente quando serviços precisam ser substituídos. Durante o processo de substituição de serviços, percebe-se que através de SOA encontram-se serviços com a mesma interface que poderiam substituir de forma transparente o serviço inativo, porém diferenças nas políticas de segurança acabam se tornando um problema.

Utilizando os padrões e especificações estudados neste capítulo procurou-se uma forma

de facilitar a implementação de criptografia em serviços que compõem arquiteturas SOA. O próximo capítulo apresenta o problema identificado, bem como a solução encontrada. Ainda é apresentado um modelo e um protótipo do mesmo.

3. SEGURANÇA BASEADA EM PROXY

Este capítulo apresenta mais detalhadamente o problema a ser solucionado, aplicando os conceitos fundamentados no Capítulo 2. A Seção 3.1 descreve o problema encontrado, e ilustra as suas formas de ocorrência visando encontrar uma solução. A Seção 3.2 apresenta a solução encontrada e a propõe através de um modelo UML. A Seção 3.3 apresenta um protótipo do modelo discutido na Seção 3.2, com o objetivo de testar a sua composição, capacidades e limitações.

3.1 Problema

Esta seção apresenta as dificuldades encontradas ao tratar de criptografia XML em serviços de aplicações baseadas em SOA. Conforme o problema também abordado em [JZ08], a variedade de algoritmos de criptografia pode tornar o desenvolvimento de aplicações compostas por vários serviços uma tarefa complexa. Como os serviços que compõem uma aplicação geralmente são providos por terceiros, as vezes é necessário que um serviço seja substituído por outro devido a vários motivos, como por exemplo desativação do serviço ou alto custo. Caso um dos serviços originais de uma aplicação SOA fique indisponível, a sua substituição pode ser uma tarefa relativamente complexa, principalmente se o serviço substituto utilizar algoritmos de criptografia diferentes. Mesmo possuindo interfaces compatíveis, é necessário recodificar a aplicação cliente para tornar possível a comunicação através do novo algoritmo de criptografia.

A Figura 3.1 ilustra a situação onde um dos serviços de uma aplicação fica indisponível e é necessário substituí-lo por outro. Dentro do repositório UDDI, temos cadastradas descrições de serviços disponibilizados por duas empresas A e B. Ambas oferecem serviços similares, S1, S2 e S3. A aplicação representada utiliza inicialmente os serviços providos por A, que em suas políticas de segurança prevê o uso do algoritmo de criptografia RSA para criptografar os arquivos XML transmitidos pela rede. Porém, a empresa A está descontinuando o serviço S1 e a aplicação passará a utilizar o serviço S1 da empresa B, que utiliza DSA. Esta nova conexão está representada pela linha azul na figura. Com base nestes dados, será necessário recodificar parte da aplicação para que se torne compatível com o novo padrão de criptografia.

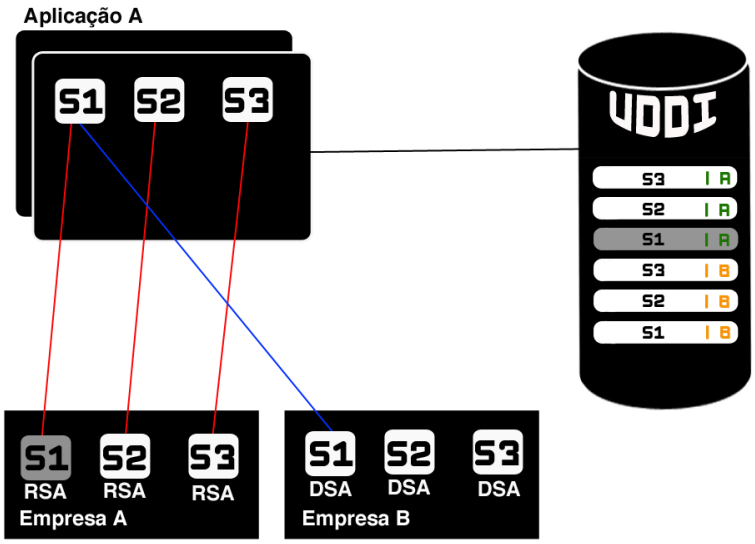

Figura 3.1: Ilustração da substituição de um serviço.

Nas Figuras 3.2 e 3.3, mostramos um exemplo com parte do arquivo de descrição da interface (WSDL) de dois serviços similares. Ambos estão cadastrados em um repositório UDDI e o desenvolvedor da aplicação é livre para escolher qual deseja. Ambos os serviços estão de acordo com *WS-Policy*, que descreve os requisitos de segurança em conjunto com a interface. Podemos visualizar entre as linhas 1 e 18 das Figuras 3.2 e 3.3 que as interfaces são exatamente iguais - utilizam o mesmo tipo de inteiro - porém, nas linhas 24 das Figuras 3.2 e 3.3 percebe-se a diferença quanto aos algoritmos de criptografia. Apesar de ambos os serviços usarem RSA, eles não são compatíveis, pois os algoritmos simétricos que criptografam as chaves não são iguais. Conforme exemplificado pela Figura 3.1, mesmo sendo compatíveis, a substituição destes serviços pode não ser trivial.

3.2 Modelo

Com base nas necessidades previamente identificadas durante o processo de pesquisa e apresentado nos capítulos anteriores, foi desenvolvido um modelo capaz de apresentar uma solução para o problema proposto na Seção 3.1.

O propósito do modelo é tratar de situações como a exemplificada na Figura 3.4a, através do qual um sistema cliente (C) acessa diferentes serviços (S1 e S2), porém, cada serviço é provido por diferentes empresas e em suas políticas são exigidos determinados algoritmos de criptografia. Para que seja possível tratar estes diferentes algoritmos, é necessário que o código da aplicação cliente preveja-os no momento do desenvolvimento. A

```
<wsdl:types>
    <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified">
        <xs:element name="calculateFibonacci">
            <xs:complexType>
                <xs:sequence>
                    <xs:element minOccurs="0" name="num" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="calculateFibonacciResponse">
            <xs:complexType>
                <xs:sequence>
                    <xs:element minOccurs="0" name="return" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:schema>
</wsdl:types>

<wsp:UsingPolicy wsdl:Required="true"/>
<wsp:Policy wsu:Id="MyPolicy">
    <wsp:ExactlyOne>
    <wsp:All>
        <sp:TripleDesRsa15 />
        <sp:EncryptedParts>
            <sp:Body/>
        </sp:EncryptedParts>
    </wsp:ExactlyOne>
</wsp:Policy>

```

Figura 3.2: WSDL com descrição WS-Policy com algoritmo *TripleDesRsa15*.

```
<wsdl:types>
    <xs:schema attributeFormDefault="qualified" elementFormDefault="qualified" >
        <xs:element name="calculateFibonacci">
            <xs:complexType>
                <xs:sequence>
                    <xs:element minOccurs="0" name="num" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="calculateFibonacciResponse">
            <xs:complexType>
                <xs:sequence>
                    <xs:element minOccurs="0" name="return" type="xs:int"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:schema>
</wsdl:types>

<wsp:UsingPolicy wsdl:Required="true"/>
<wsp:Policy wsu:Id="MyPolicy">
    <wsp:ExactlyOne>
    <wsp:All>
        <sp:Basic256Rsa15 />
        <sp:EncryptedParts>
            <sp:Body/>
        </sp:EncryptedParts>
    </wsp:ExactlyOne>
</wsp:Policy>

```

Figura 3.3: WSDL com descrição WS-Policy com algoritmo *Basic256Rsa15*.

situação fica complicada no momento em que um serviço deve ser substituído por um novo. Neste caso, S2 deve ser substituído por S3. Como vemos, as suas políticas de segurança são diferentes e o cliente necessita ser recodificado (com o algoritmo x, e não mais com o DSA) para tornar a comunicação com S3 possível.

O modelo propõe que a estrutura seja conforme a Figura 3.4b, onde há um intermediário. Considerando esta nova estrutura, o sistema cliente pode ser codificado pensando apenas em 1 ou em nenhum algoritmo de criptografia, o intermediário se torna o responsável por criptografar os dados com outros algoritmos, de modo a facilitar a codificação e a manutenção da aplicação cliente.

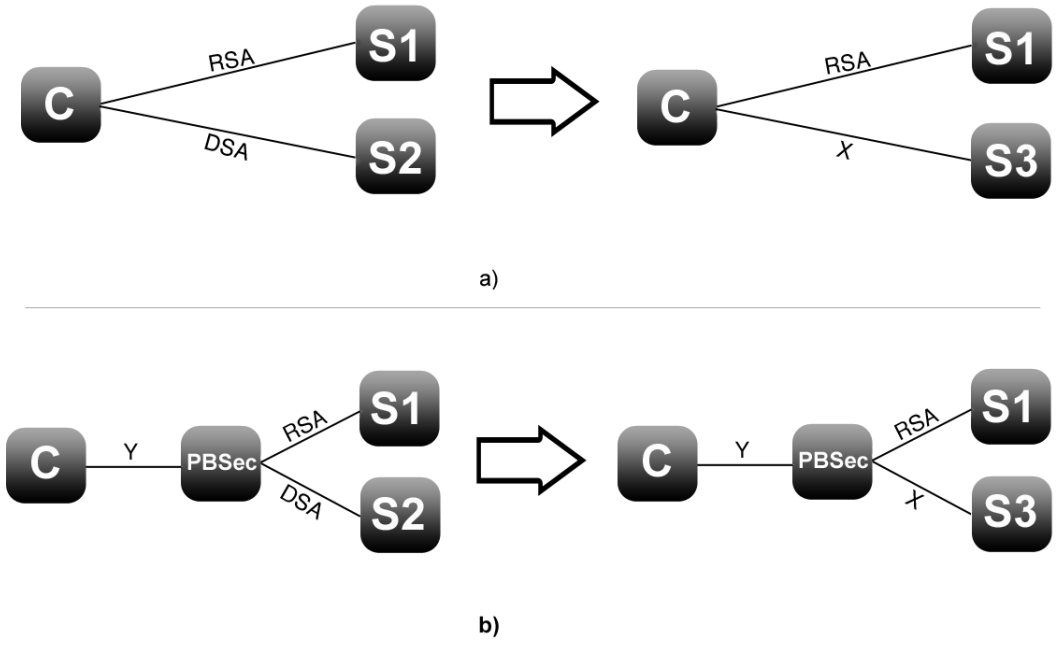

Figura 3.4: Propósito do Modelo

O sistema implementado com base no modelo deve ser capaz de receber as mensagens de um serviço, decodificá-las e recodificá-las utilizando o algoritmo de criptografia apropriado e, então, encaminhá-las para o serviço destinatário.

Conforme a Figura 3.5, o modelo do sistema prevê alguns casos de uso através do qual é permitido ao usuário: fazer login, listar, excluir e cadastrar serviços.

Cada Caso de Uso é descrito abaixo:

- FazLogin: está previsto que o cliente deve ser cadastrado no sistema com *login* e senha desta forma, o sistema pode confirmar a identidade do usuário e relacionar os serviços com o seu perfil. Sugere-se a utilização de certificados X.509 para a autenticação dos usuários (Tabela 3.1).

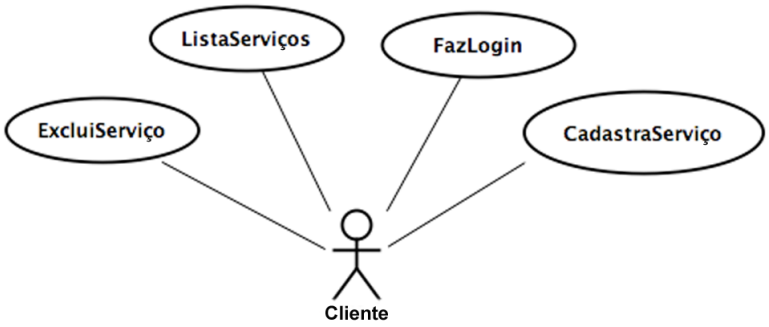

Figura 3.5: Casos de Uso do Sistema

Tabela 3.1: Caso de Uso FazLogin

Característica	Descrição
Ator Primário	Usuário
Objetivo	Permitir ao usuário cadastrado no sistema acessar as suas funcionalidades.
Fluxo Principal	1. O usuário acessa a página do serviço. 2. Usuário insere *login* e senha. 3. O sistema criptografa os dados de *login* 4. O sistema realiza a validação do *login* e senha. 5. O sistema retorna o menu principal do serviço.
Fluxo Alternativo	4a. *Login* e senha não são válidos. 4b. O Sistema exibe a mensagem "*Login* ou senha Inválidos". 4c. Sistema retorna ao passo 2.

- ListaServiços: o usuário pode listar todos os seus serviços cadastrados. Desta forma o sistema consulta o banco de dados buscando as informações dos serviços previamente listados (Tabela 3.2).

Tabela 3.2: Caso de Uso ListaServiços

Característica	Descrição
Ator Primário	Usuário
Objetivo	Permitir ao usuário verificar serviços cadastrados previamente.
Fluxo Principal	1. O usuário acessa no menu a opção Listar Serviços. 2. O sistema busca a lista de serviços no banco de dados. 3. O sistema exibe os serviços encontrados.
Fluxo Alternativo	3a. Detalhes dos Serviços. 3b. O usuário pode optar por visualizar mais detalhes de determinado serviço.

- ExcluiServiços: entre os serviços listados, este caso de uso prevê a exclusão de qualquer um dos serviços, eliminando as suas possibilidades de uso (Tabela 3.3).
- CadastraServiço: é o responsável por interpretar o arquivo WSDL que descreve o serviço, gerar as chaves de criptografia corretas e gerar o código do sistema intermediário (Tabela 3.4).

Tabela 3.3: Caso de Uso ExcluiServiços

Característica	Descrição
Ator Primário	Usuário
Objetivo	Permitir ao usuário excluir serviços cadastrados no sistema
Fluxo Principal	1. O usuário acessa no menu a opção Excluir Serviço. 2. O sistema busca a lista de serviços no banco de dados. 3. O usuário seleciona o serviço que deve ser excluído. 4. O sistema verifica se não há ninguém utilizando o serviço. 5. O sistema exclui o serviço.
Fluxo Alternativo	4a. Serviço não pode ser excluído. 4b. O Sistema exibe a mensagem "Serviço em Uso, não foi possível excluir o serviço". 4c. Sistema retorna ao passo 2.

Tabela 3.4: Caso de Uso CadastraServiço

Característica	Descrição
Ator Primário	Usuário
Objetivo	Permitir ao usuário cadastre novos serviços no sistema
Fluxo Principal	1. O usuário acessa no menu a opção Cadastrar Serviço. 2. O sistema disponibiliza um campo para o upload do arquivo WSDL 3. O usuário efetua o upload do arquivo. 4. O sistema interpreta o arquivo. 5. O sistema valida os dados. 6. O sistema gera o novo serviço. 7. O sistema insere o serviço na lista de serviços. 8. O sistema retorna os detalhes do serviço.
Fluxo Alternativo	4a. Os dados do arquivo WSDL não são válidos. 4b. O Sistema exibe a mensagem "Dados Inválidos". 4c. Sistema retorna ao passo 2.

A Figura 3.6 representa, através de um diagrama de sequência, como o usuário interage com o sistema durante o processo de Cadastro de Serviço. Primeiramente é necessário que o usuário efetue o *login*; após o sistema autenticar o usuário, se torna disponível a opção de cadastro de serviços; para cadastrar um serviço, é necessário efetuar o *upload* do arquivo WSDL contendo as especificações de comunicação e de segurança (*WS-Policy*) do serviço.

Após efetuado o *upload* do serviço, é realizada a interpretação do arquivo para que sejam retiradas as informações necessárias para a geração do sistema de *proxy*. Desta forma, o sistema pode ser gerado corretamente; uma vez com o sistema gerado, é devolvido ao usuário o caminho para que o serviço seja utilizado.

A Figura 3.7, apresenta através de um diagrama de sequência, o processo de comunicação completo entre um cliente e um servidor, tendo o sistema de *proxy* como intermediário. Primeiramente o usuário solicita a chave ao sistema intermediário, este por sua vez devolve a chave. Com a chave obtida o sistema pode criptografar a mensagem, e enviá-la ao intermediário que vai decriptografar e recriptografar utilizando a chave obtida a partir do serviço final, enviando a mensagem para que o serviço possa analisá-la e calcular o resultado.

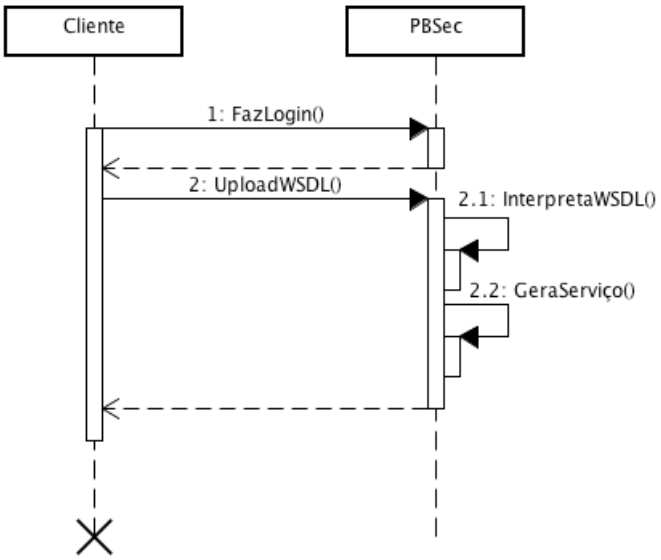

Figura 3.6: Diagrama de Sequência do Cadastramento de Serviços

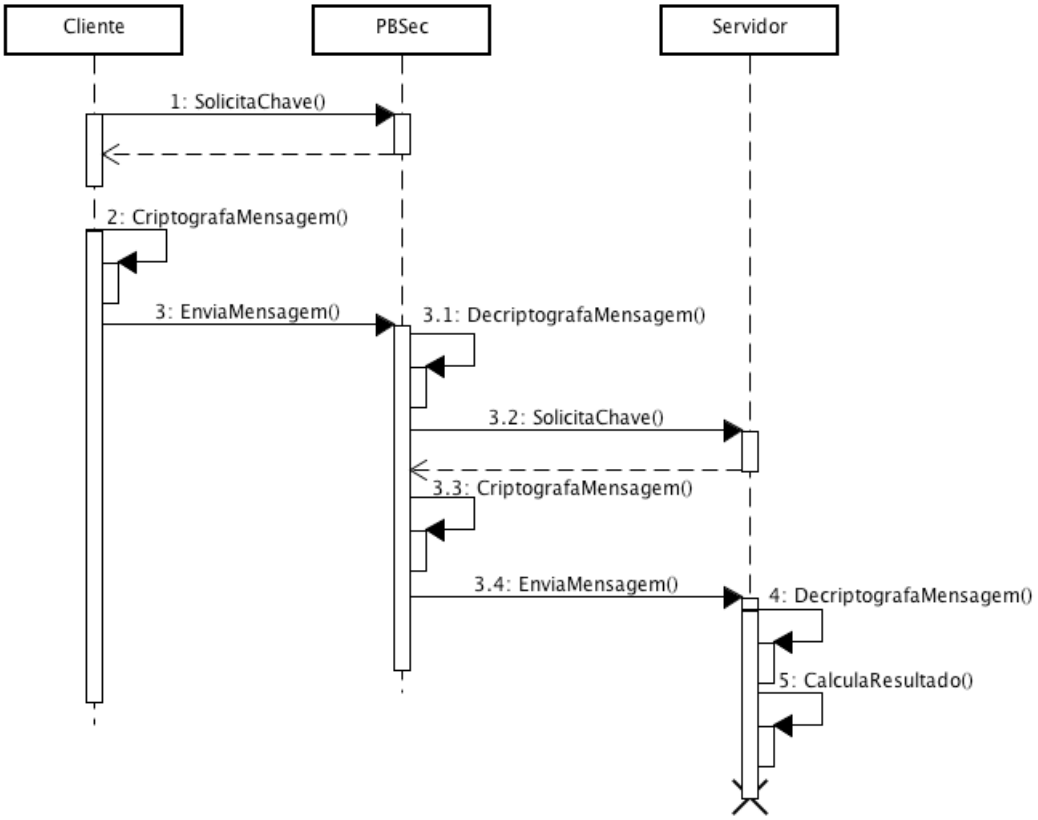

Figura 3.7: Diagrama de Sequência da comunicação com o *proxy*

3.3 Protótipo

O protótipo é baseado no padrão *Web Services* provido pela W3C [HBN+04] utilizando, portanto, seus principais conceitos: XML [New02], SOAP [BEK+00], WSDL [CCMW01] e UDDI [ACKM03]. Para este último, o próprio sistema atua como um repositório de cadastro e pesquisa.

O sistema foi desenvolvido em módulos e está organizado conforme a Figura 3.8. O núcleo é responsável exclusivamente pela comunicação entre serviços; a Interface Web permite o cadastro de serviços no núcleo através da interpretação dos arquivos WSDL; o módulo de segurança é responsável por armazenar e gerenciar as chaves criptográficas; o banco de dados é o responsável por manter as informações dos serviços, bem como usuários e senhas.

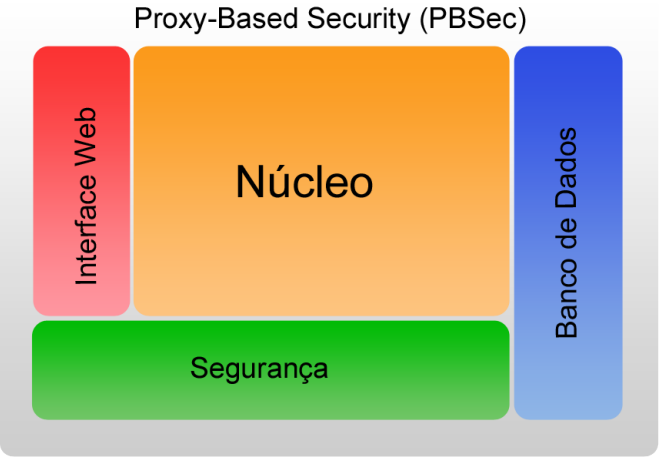

Figura 3.8: Módulos do Sistema Protótipo

A implementação foi realizada através da utilização das linguagens Java (JDK 1.5.0_17) para o núcleo do sistema e PHP (PHP5) para o acesso *Web*, com banco de dados MySQL. Os *frameworks Web Services* escolhidos para o sistema foram: Axis2 [Apa09a] para a comunicação e Rampart [Apa08] para o módulo de segurança. Para o acesso *Web*, foi utilizado o *framework* PHP *NuSoap*, o qual é o responsável pela interpretação dos arquivos WSDL e *WS-Policy*. Para a geração e gerenciamento das chaves de criptografia foi utilizado o aplicativo *BouncyCastle* [Bou09]. O servidor onde as aplicações executam é o TomCat 6.0 [Apa09b] do projeto Apache.

O Axis2 versão 1.4.1 é uma *engine* para *Web Services*, baseada na linguagem de programação Java, porém está disponível também para C. Este *framework* prevê que as interfaces WSDL sejam adicionadas às aplicações, servindo também como um servidor de aplicações. Para a utilização da especificação *WS-Security* [O'N03] foi utilizado o módulo

Rampart, pois este facilita a implementação das especificações WSS, uma vez que possui os algoritmos já implementados.

O *BouncyCastle* é um conjunto de APIs utilizadas para criptografia. Ele implementa vários algoritmos de criptografia, conforme a Tabela 3.5. Através do *BouncyCastle* é possível gerar e gerenciar as chaves de criptografia utilizando o *keytool*, disponível na JDK através das bibliotecas JCE (*Java Cryptography Extension*) [Sun09]. As chaves ficam armazenadas no formato PKCS12 (*Public Key Cryptography Standards #12*) [Lab99] e os algoritmos suportados estão na Tabela 3.5.

Tabela 3.5: Algoritmos de criptografia suportados pelo BouncyCastle

Algoritmos Assimétricos	Algoritmo de Assinatura
DSA	SHA1
RSA	RSA
	MD2
	MD5
	SHA256
	SHA238
	SHA512
ECDSA	SHA1
	SHA224
	SHA256
	SHA384
	SHA512

3.3.1 Núcleo

O núcleo da aplicação é responsável pelo gerenciamento e compilação do código gerado pela Interface Web. O processo de compilação é realizado através da aplicação Apache Ant[1], que esquematiza todos os passos de compilação em um arquivo XML, de forma similar a um *script bat* em computadores com sistema operacional Windows.

Ao compilar os arquivos *.java* gerados e alterados pela *Interface Web*, o núcleo organiza os serviços de acordo com o usuário e com o serviço, criando uma hierarquia organizacional que facilita uma possível manutenção dos serviços.

Para a publicação dos serviços é necessário efetuar a geração de um pacote *.aar* com os arquivos .java compilados. Este pacote é muito similar aos pacotes .jar, uma vez que são compostos por pastas e arquivos necessários para que o serviço seja executado corretamente. Os arquivos de configuração utilizados na geração do pacote com os serviços, bem como o *keystore* que armazena as chaves, podem ser vistos em maiores detalhes no Apêndice A.

[1] O Apache Ant pode ser rodado a partir da Web através da função PHP “exec".

3.3.2 Interface Web

A Interface Web é responsável por interpretar os arquivos WSDL, gravar as informações no banco de dados e gerar os arquivos necessários para o funcionamento do serviço no núcleo. A Figura 3.9 ilustra a página inicial da aplicação logo após o *login*. Conforme previsto no modelo é possível listar e cadastrar serviços. Para a opção listar serviços é realizada uma busca no banco de dados, exibindo os serviços cadastrados. Na opção cadastrar é exigido do usuário que faça o *upload* do arquivo de descrição do serviço, o arquivo é interpretado automaticamente e executa as chamadas ao banco e a geração dos arquivos.

Figura 3.9: Tela Inicial PBSec

A Figura 3.10 apresenta o exemplo de um arquivo WSDL interpretado, exibindo na tela as suas informações básicas. O serviço utilizado como exemplo é "*FibonacciService*", sendo que este é capaz de calcular um determinado termo de uma sequência Fibonacci. Neste caso, o arquivo WSDL foi enviado, tendo sido identificado apenas 1 método, o "*calculateFibonacci*", o qual apresenta também o caminho do serviço no campo "*Endpoint*", entre outras informações importantes para a geração do serviço intermediário.

3.3.3 Banco de Dados

A estrutura do Banco de Dados que compõe o sistema PBSec é relativamente simples. Ela é composta por 3 Tabelas responsáveis por armazenar os usuários, os serviços e as empresas que disponibilizam os serviços. Os dados do banco são criptografados com a função *hash* SHA1, a qual garante a segurança das senhas e dos dados de acesso, enquanto elas estiverem armazenadas.

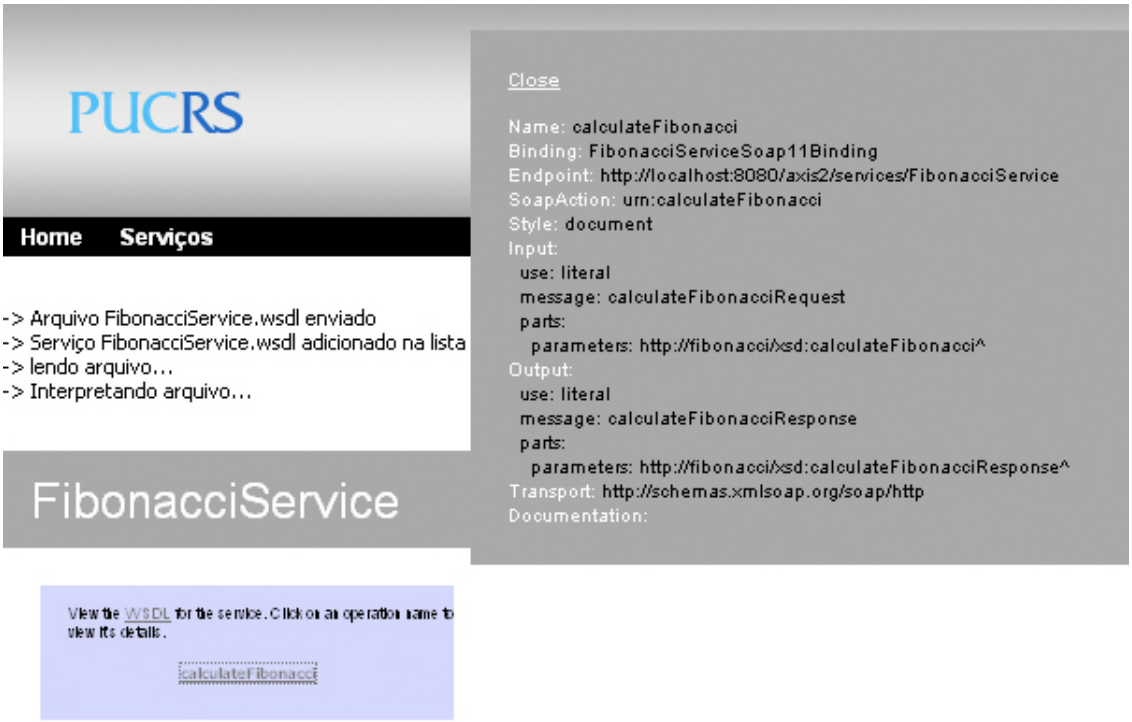

Figura 3.10: Tela de Cadastro com os dados do arquivo WSDL analisado

A Figura 3.11 apresenta a estrutura do banco de dados do sistema PBSec. A Tabela *Users*, responsável por identificar os usuários, possui um campo ID, que é um identificador único do usuário e serve para relacionar os usuários com os serviços que ele cadastrou. Ainda há os campos *Nome* e *Sobrenome* que identificam o usuário e também os campos *Login* e *Passwrd* responsáveis por garantir que apenas o usuário certo irá utilizar o serviço. Na Tabela *Serviços* há o campo ID que identifica os serviços de forma única e os relaciona com os usuários e com as empresas. Os demais campos contêm os dados retirados do arquivo de descrição WSDL e também o caminho de acesso ao serviço, todas informações importantes para a utilização do sistemas PBSec. Já na Tabela *Empresas*, existem os dados das empresas que disponibilizam os serviços, como por exemplo, formas de contato.

3.3.4 Segurança

Por ser um sistema de segurança, as suas especificações devem ser garantia de confiabilidade para os clientes Estas especificações abrangem todo o sistema, desde a *Interface Web* até o Banco de Dados.

Como mencionado nas Subseções 3.3.1, 3.3.2 e 3.3.3, existem características específicas para cada módulo do sistema. A *Interface Web* é disponibilizada através do protocolo HTTPS, com certificado SSL emitido pela certificadora *Verisign* [Ver09]. Além disso, cada

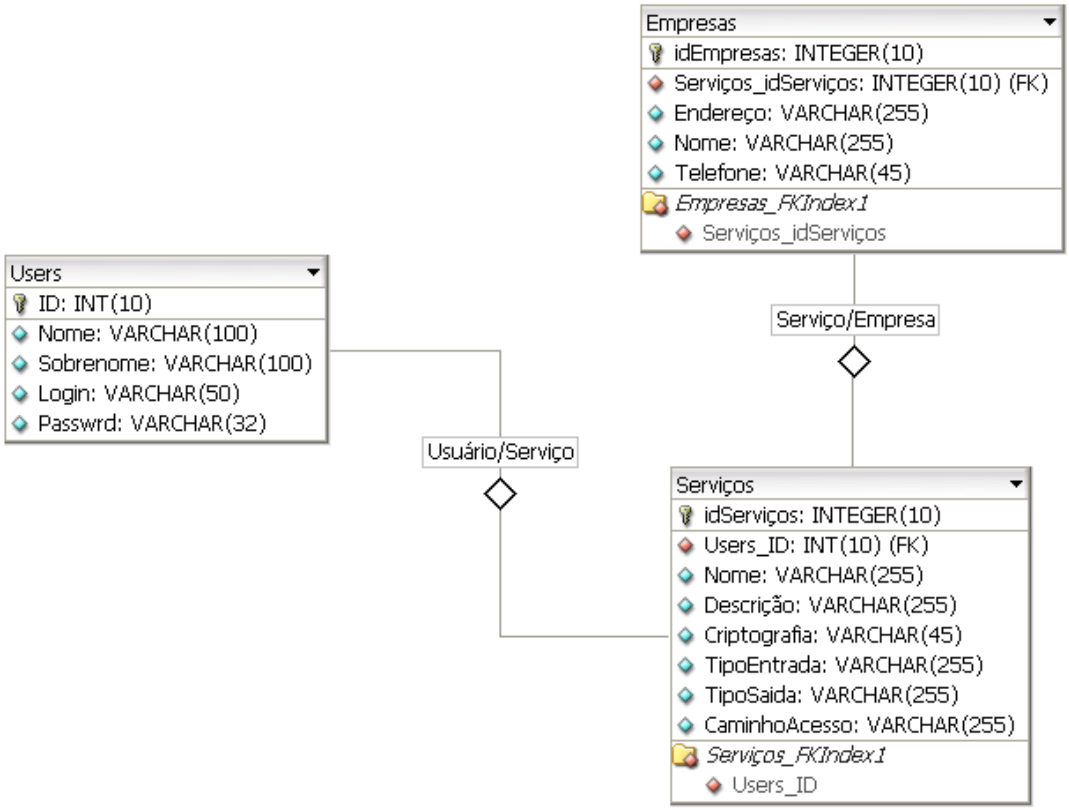

Figura 3.11: Estrutura do Banco de Dados

usuário possui dados identificação individuais, os quais são responsáveis por autenticar os usuários.

O núcleo do sistema possui proteção de variáveis, protegendo as informações que transitam pelo sistema mesmo quando elas estejam armazenadas na memória local, uma vez que é necessário decriptografar os dados e criptografar novamente. Com o auxílio do aplicativo *BouncyCastle* e da biblioteca JCE, o núcleo e o módulo de segurança gerenciam todas as chaves dos serviços que utilizam o PBSec. As chaves criptográficas ficam armazenadas em um formato seguro, chamado PKCS12, o qual é um padrão para armazenamento de chaves amplamente difundido.

3.3.5 Limitações do Protótipo

Este protótipo está preparado para tratar apenas Criptografia XML. A forma de geração dos arquivos intermediários não é capaz de contornar outras especificações de segurança, por exemplo autenticação. Para contornar estas configurações é necessário corrigir o código manualmente. Seria possível contornar esta deficiência utilizando um codificador automático, como o WSDL2Java contido no *framework* Axis2 [Apa09a].

As limitações do protótipo dificultam o uso em algumas aplicações reais, pois normalmente existem várias outras especificações de segurança e orquestração suplementares, que podem tornar o sistema inoperante.

Outra limitação identificada é que as chaves criptográficas necessitam ser cadastradas no sistema manualmente através de um campo de *upload* disponível na página. Esta troca poderia ser automática, o que torna o processo mais seguro e mais simples para o usuário.

Com base no protótipo apresentado neste capítulo, estamos aptos a demonstrar a efetividade do modelo através da sua aplicação em um estudo de caso. O Capítulo 4 apresenta um estudo de caso que utiliza aplicações reais e simula o sistema de uma agência de turismo, onde o sistema é composto por serviços providos por diferentes empresas, como companhias aéreas e, portanto, possuem diferentes pré-requisitos de segurança.

4. ESTUDO DE CASO

Para testar como um serviço de segurança baseado em *proxy* pode ser utilizado para construir aplicações SOA de uma maneira mais flexível, nós escolhemos como exemplo o sistema de uma agência de viagens, conforme já utilizado como estudo de caso em [ZPR03]. Este é um sistema típico e apresenta as principais características encontradas em sistemas reais.

Um sistema de agência de viagens normalmente é composto por vários serviços, como por exemplo, cruzeiros marítimos, passagens aéreas, hotéis e restaurantes. Cada um destes serviços é provido por diferentes empresas. No caso das passagens aéreas, podemos ter um serviço disponibilizado pela TAM, outro pela GOL, outro pela AZUL e assim sucessivamente. Esta é, portanto, uma aplicação que tradicionalmente pode caracterizar uma SOA, pois os serviços que compõem a aplicação são disponibilizados por diferentes empresas através de um repositório e de interfaces bem definidas.

A Figura 4.1 representa como seria a estrutura desta aplicação. Cada número representa uma empresa e cada cor um tipo de serviço: os azuis são companhias aéreas, os pretos restaurantes, os verdes hotéis e os laranjas companhias marítimas. Como cada serviço é disponibilizado por uma empresa, eles possuem políticas de segurança diferentes, uma vez que não há um acordo formal entre as empresas. Apesar disso, todas as interfaces são descritas com WSDL e as políticas de segurança com *WS-Policy* seguindo o padrão Web Service. Esta figura ilustra uma implementação similar existente no mercado, na qual a implementação apresentada na Seção 4.1 será baseada.

As principais dificuldades em desenvolver uma aplicação baseada em SOA são a confiabilidade dos serviços e as diferentes políticas de segurança. Relacionado diretamente com o nosso trabalho, devemos lidar com as diferentes tecnologias empregadas para proteção de dados, vários protocolos de criptografia, vários *tokens* de autenticação, entre outras. Cada empresa é livre para escolher o que deseja exigir para a utilização do seu serviço. Por exemplo, para integrar aplicações utilizando diferentes algoritmos para Criptografia XML é necessário tratar de cada algoritmo separadamente, gerando uma maior codificação.

O objetivo deste estudo de caso é testar a usabilidade do protótipo PBSec em situações reais, de modo a deixar clara a dificuldade em seguir as diferentes políticas de segurança aplicadas por diferentes empresas provedoras de serviços. Pretende-se, através deste es-

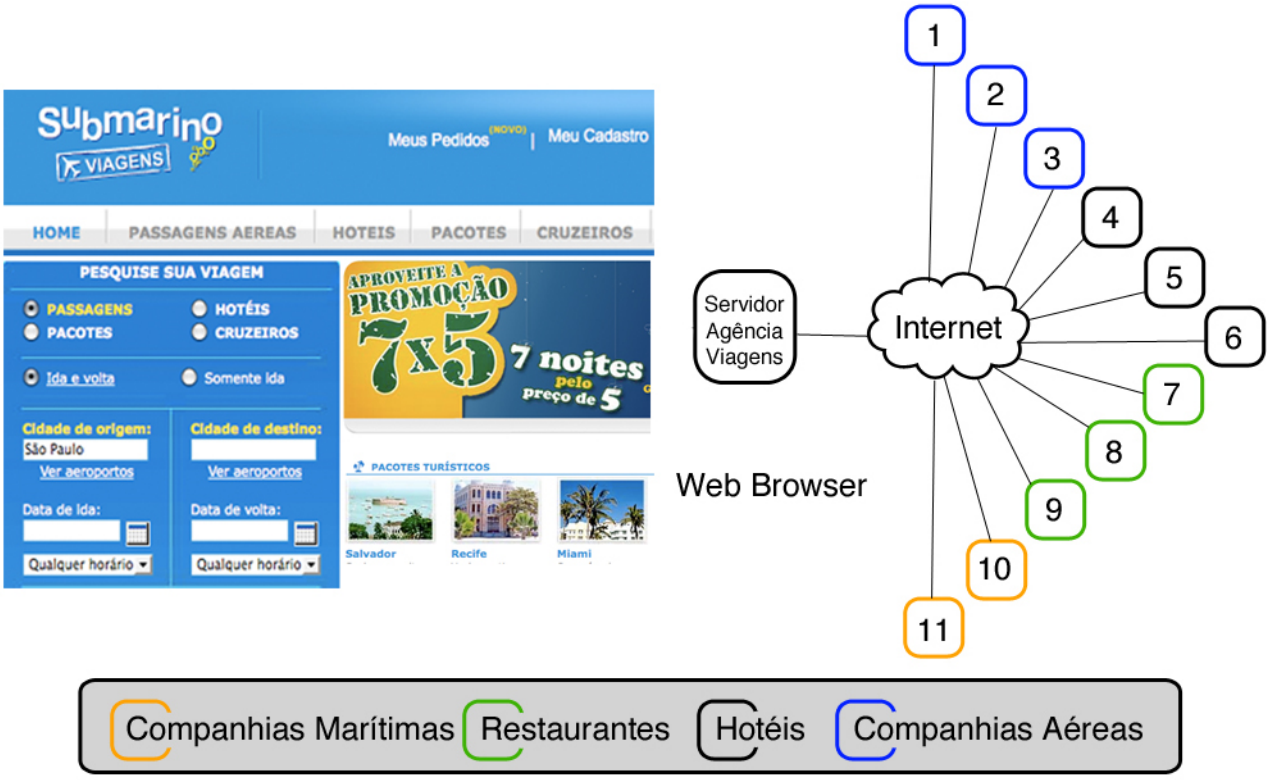

Figura 4.1: Ilustração do Funcionamento de uma Agência de Viagens

tudo de caso, apresentar uma forma mais simples de se prover segurança a aplicações que seguem o padrão SOA.

4.1 Implementação

A implementação do estudo de caso é composta por 3 serviços: um serviço (reserva de quartos) sem criptografia, disponível na Internet, um serviço (pagamento) com outras formas de segurança, como autenticação, disponível na Internet e um serviço com criptografia (reserva de passagens aéreas) desenvolvido para testes juntamente com o protótipo. Estes serviços são providos pelas seguintes empresas:

- Paypal: Fornece o seu serviço de pagamento também em forma de Web Service. Este serviço faz parte da API de testes *SandBox* [Pay09], a qual não tem valor real. É possível acessar a sua descrição em:

  https://www.paypal.com/wsdl/paypalsvc.wsdl

- Hotel.de: O site europeu Hotel.de fornece o serviço de reserva de quartos para uma variedade de hotéis em diferentes cidades e países europeus. A descrição do seu serviço está disponível em:

http://publicwebservices.hotel.de/V2_7/FreeHotelSearchWebService.asmx?wsdl

- Próprio: O Serviço de passagens aéreas foi desenvolvido em conjunto com o PBSec e é disponibilizado localmente na rede de testes.

A aplicação foi implementada em PHP para que pudesse ser utilizado através de um navegador Web, conforme a maioria dos portais de agencias de viagens. Através desta página, o usuário é capaz de verificar a disponibilidade de hotéis e voos, adicionando ao seu "carrinho de compras" os destinos desejados. Após selecionar todos os dados necessários, é possível efetuar o pagamento, condicionalmente, se o pagamento for confirmado são liberadas as reservas.

4.1.1 Cadastro dos Serviços

Para a utilização do PBSec, o primeiro passo é cadastrar os serviços desejados no sistema. O cadastramento de serviços é relativamente simples, uma vez que se baseia principalmente no *upload* do arquivo de descrição de interfaces *wsdl*. Com os serviços implementados podemos cadastrá-los no PBSec.

O primeiro serviço a ser castrado é o responsável pelos hotéis. A Figura 4.2 mostra a tela de retorno do sistema, imediatamente após o serviço ter seu arquivo de descrição de interfaces enviado e processado pelo sistema. Ainda podemos ver na Figura 4.2 que o serviço de reserva de quartos de hotéis se chama "*FreeHotelSearchWebService*" e é composto por 14 métodos, os quais permitem ao usuário buscar hotéis disponíveis para uma determinada cidade, verificar a disponibilidade de quartos para cada hotel, entre outros métodos.

O segundo serviço cadastrado foi o serviço responsável pelos pagamentos. A Figura 4.3 apresenta o retorno dos dados que o PBSec extraiu do arquivo de descrição referente a este serviço. O serviço disponibilizado pela PayPal originalmente se chama "*PayPalAPIInterfaceService*" e é composto por 48 métodos, sendo que apenas alguns destes aparecem listados na figura. Entre as ações possíveis está efetuar pagamentos, pesquisar por pagamentos efetuados anteriormente, entre outros.

O último serviço a ser cadastrado foi o responsável pela reserva de passagens aéreas. Da mesma forma como nas figuras anteriores, a Figura 4.4 apresenta os dados do arquivo de descrição de interfaces. O serviço originalmente se chama "*FlightCompany*" e é uma simplificação de um serviço real, sendo que este é composto por 3 métodos que permitem a busca por passagens, a busca pelos valores e a reserva das mesmas.

Uma vez com todos os serviços devidamente cadastrados e com as suas particularidades consideradas, a próxima seção apresenta os testes efetuados e os resultados encontrados.

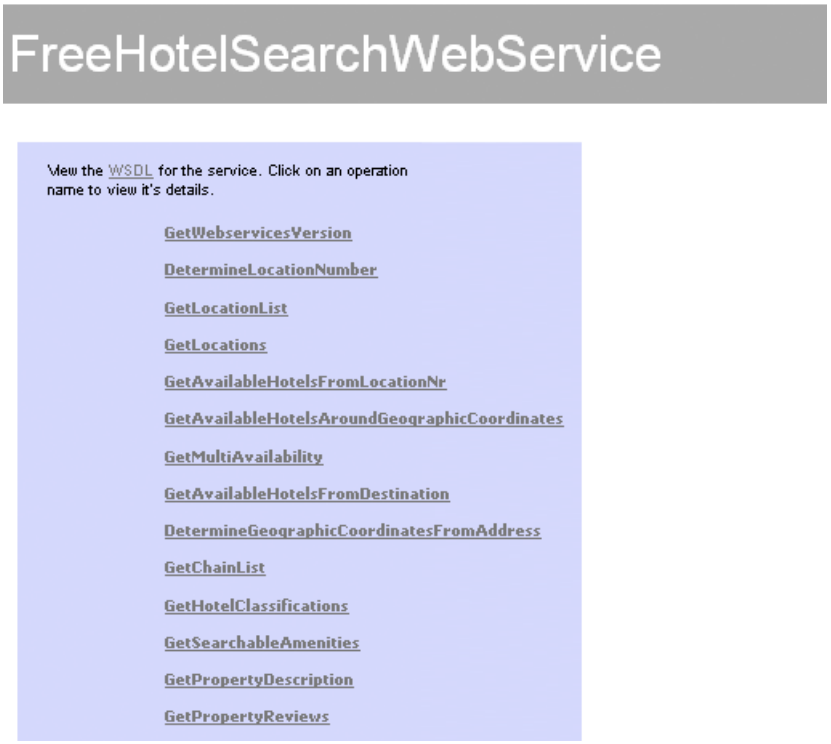

Figura 4.2: Cadastro do serviço Hotel.de

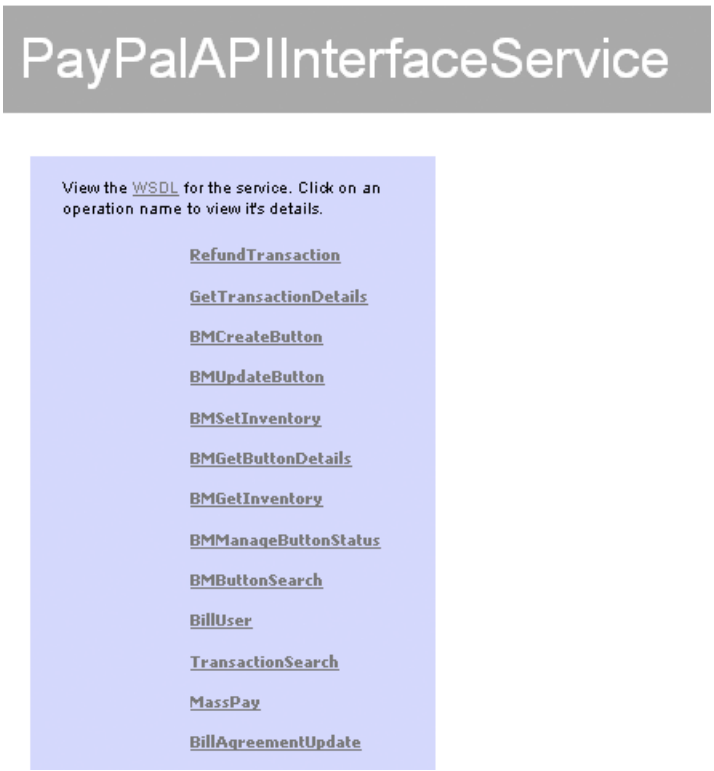

Figura 4.3: Cadastro do Serviço PayPal

Figura 4.4: Cadastro do Serviço de Passagens Aéreas

4.2 Resultados

Os resultados apresentados a seguir são baseados em testes realizados a partir da implementação do estudo de caso e tem como objetivo analisar o *overhead* gerado pela inclusão do PBSec. Os serviços foram executados considerando a infra-estrutura apresentada na Figura 4.5. Considera-se como a primeira parte da conexão, o trecho entre o cliente e o sistema PBSec; consequentemente a parte entre o PBSec e o serviço final será chamada de segunda parte.

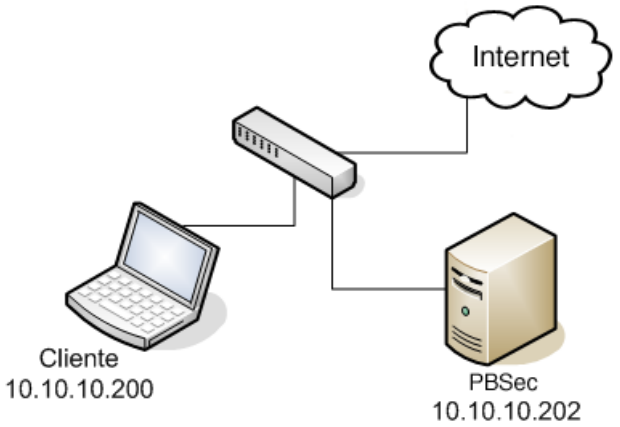

Figura 4.5: Rede de testes do estudo de caso

Para garantir a confiabilidade dos resultados foi utilizada a técnica de amostragem sim-

ples [AKD04]. Foram realizadas 1000 execuções com cada teste. Com base nestes resultados foi calculado o desvio padrão, margem de erro, confiabilidade e tamanho da amostra. Entre os 1000 resultados, o tamanho da amostra foi escolhido aleatóriamente através de um algoritmo de sorteio de números pseudo-randômicos. Os resultados apresentados possuem uma confiabilidade de 95% e margem de erro de 5 milissegundos para mais ou para menos.

Os serviços foram testados individualmente, negociando diretamente com o serviço e utilizando o PBSec como sistema intermediário. Todos os resultados estão expressados em milissegundos.

A Figura 4.6 apresenta os testes realizados apenas entre clientes e servidores, sem o PBSec. A partir disto, podemos estabelecer um parâmetro de qual seria o tempo de resposta dos serviços caso tenham sido implementados da maneira convencional, de forma a permitir que estes dados sirvam de base para o cálculo do *overhead* gerado pelo PBSec. A Tabela 4.1 apresenta estes mesmos valores e os seus dados estatísticos, como desvio padrão, confiabilidade e margem de erro.

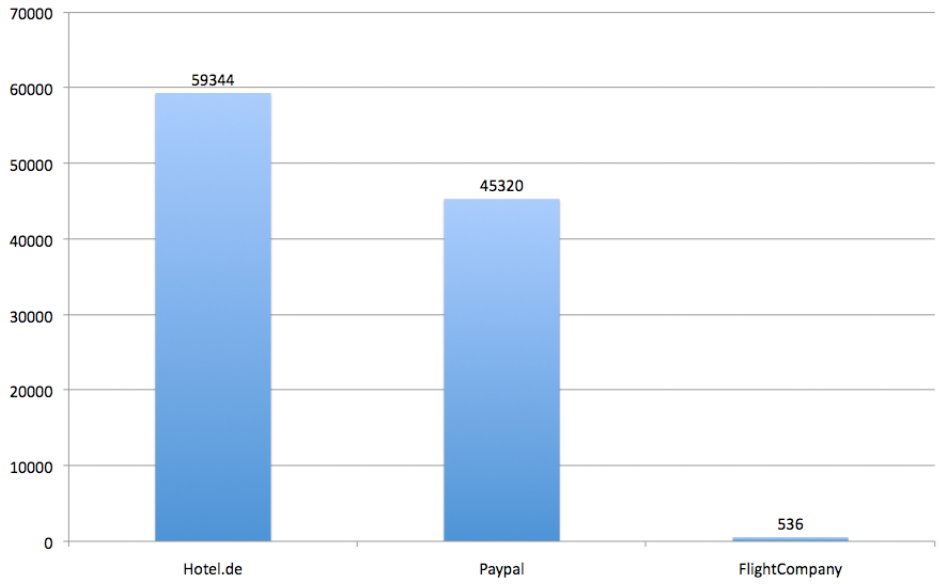

Figura 4.6: Resultado dos testes sem PBSec

Tabela 4.1: Estatísticas dos resultados apresentados na Figura 4.6

Serviço	Valor	Desvio Padrão	Confiabilidade	Margem de Erro
Hotel.de	59344	24,18	95%	5
PayPal	45320	23,82	95%	5
FlightCompany	536	6,17	95%	5

Os testes foram separados em etapas, conforme as suas variações:

- Sem Criptografia - A primeira parte da comunicação (entre o cliente e o PBSec) sem criptografia.

- Com Criptografia RSA - Com a primeira parte da comunicação com criptografia RSA de 1024 bits.

- Com Criptografia DSA - Com a primeira parte da comunicação com criptografia DSA de 1024 bits.

- Com Criptografia ECDSA - Com a primeira parte da comunicação com criptografia ECDSA de 1024 bits.

Considera-se que a segunda parte do processo de comunicação não é possível variar devido as políticas de segurança disponibilizadas por cada empresa provedora dos serviços utilizados. Também não estão sendo consideradas diferenças entre os algoritmos de criptografia, e.g. uma comunicação criptografada com algoritmo ECDSA de 512 bits seria equivalente a uma comunicação com RSA de 1024 bits. É considerado apenas o tempo de comunicação para determinado tamanho de chave criptográfica e não a efetividade da força criptográfica entre os algoritmos testados.

Testes Sem Criptografia

A Figura 4.7 ilustra os testes realizados com o sistema intermediário PBSec, considerando que a primeira parte da comunicação não utiliza criptografia. Comparando os resultados apresentados nesta figura com os apresentados na Figura 4.6, podemos calcular o *overhead* causado pelo PBSec. A Tabela 4.2 apresenta o *overhead* para os serviços sem possuir a primeira parte da conexão criptografada.

Tabela 4.2: *Overhead* Sem Criptografia na Primeira parte da Conexão

Serviço	**Tempo (ms)**
Hotel.de	110
PayPal	115
FlightCompany	120

A Tabela 4.3 apresenta os primeiros resultados utilizando o PBSec, tempo de execução, desvio padrão. Comparativamente com resultados apresentados na Tabela 4.1, podemos perceber um *overhead* baixo, devido provavelmente ao não uso de criptografia na primeira parte da conexão e também do fato do serviço PBSec estar sendo executando localmente.

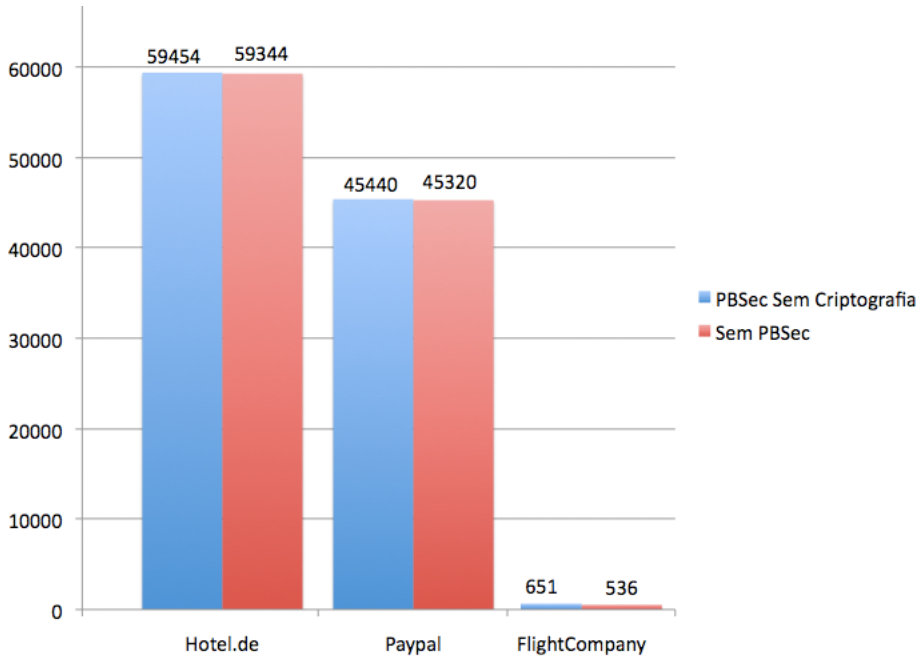

Figura 4.7: Resultado dos testes com PBSec sem criptografia na primeira parte

Tabela 4.3: Estatísticas dos resultados apresentados na Figura 4.7

Serviço	Valor	Desvio Padrão	Confiabilidade	Margem de Erro
Hotel.de	59545	24,05	95%	5
PayPal	45440	24,18	95%	5
FlightCompany	651	5,43	95%	5

Com Criptografia RSA

Os dados apresentados na Figura 4.8 ilustram os resultados de acesso aos serviços considerando o uso de criptografia RSA com chave de 1024 bits para a comunicação entre o cliente e o PBSec. Os resultados apresentados na Tabela 4.4 contêm outras informações estatísticas sobre os resultados apresentados na Figura 4.8.

Tabela 4.4: Estatísticas dos resultados apresentados na Figura 4.8

Serviço	Valor	Desvio Padrão	Confiabilidade	Margem de Erro
Hotel.de	59922	24,03	95%	5
PayPal	45905	23,76	95%	5
FlightCompany	1103	7,69	95%	5

A Tabela 4.5 demonstra o *overhead* considerando uma conexão com a primeira parte criptografada. Com base nos dados desta tabela e nos dados apresentados na Tabela 4.4, podemos perceber um aumento do tempo de comunicação em mais de 5 vezes, se comparado a conexão sem criptografia.

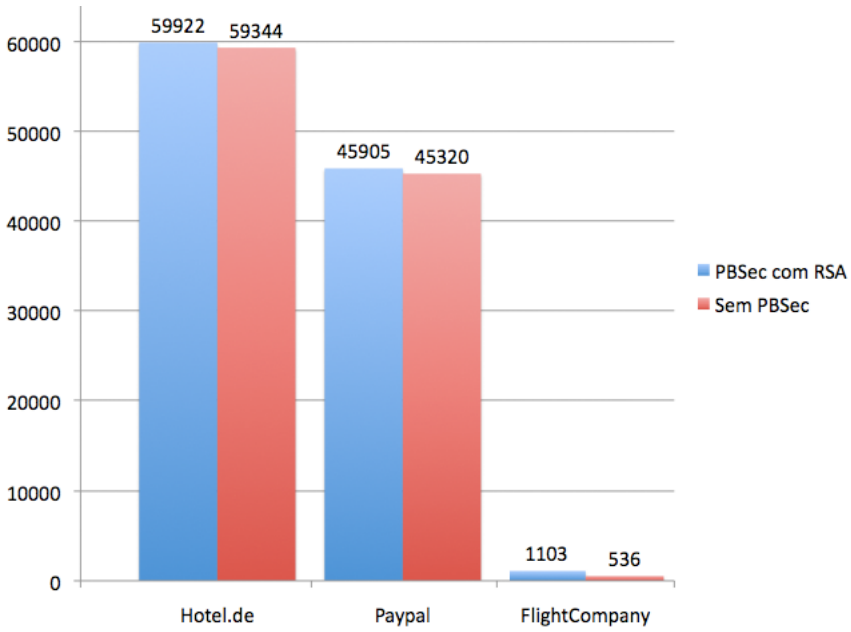

Figura 4.8: Resultado dos testes com criptografia RSA

Tabela 4.5: *Overhead* Com Criptografia RSA na Primeira parte da Conexão

Serviço	Tempo (ms)
Hotel.de	578
PayPal	585
FlightCompany	567

Com Criptografia ECDSA

A Figura 4.9 demonstra os resultados considerando o uso de criptografia ECDSA com chave de 1024 bits. A Tabela 4.6 apresenta outras estatísticas sobre os serviços.

Tabela 4.6: Estatísticas dos resultados apresentados na Figura 4.9

Serviço	Valor	Desvio Padrão	Confiabilidade	Margem de Erro
Hotel.de	59950	20,87	95%	5
PayPal	45932	20,82	95%	5
FlightCompany	1120	6,70	95%	5

A Tabela 4.7 apresenta o *overhead* da comunicação na primeira parte da conexão, utilizando criptografia ECDSA. Se comparado com os resultados apresentados na Tabela 4.5, podemos perceber uma diferença de aproximadamente 30 milissegundos, o que ressalta a diferença entre tempos criptográficos entre diferentes algoritmos.

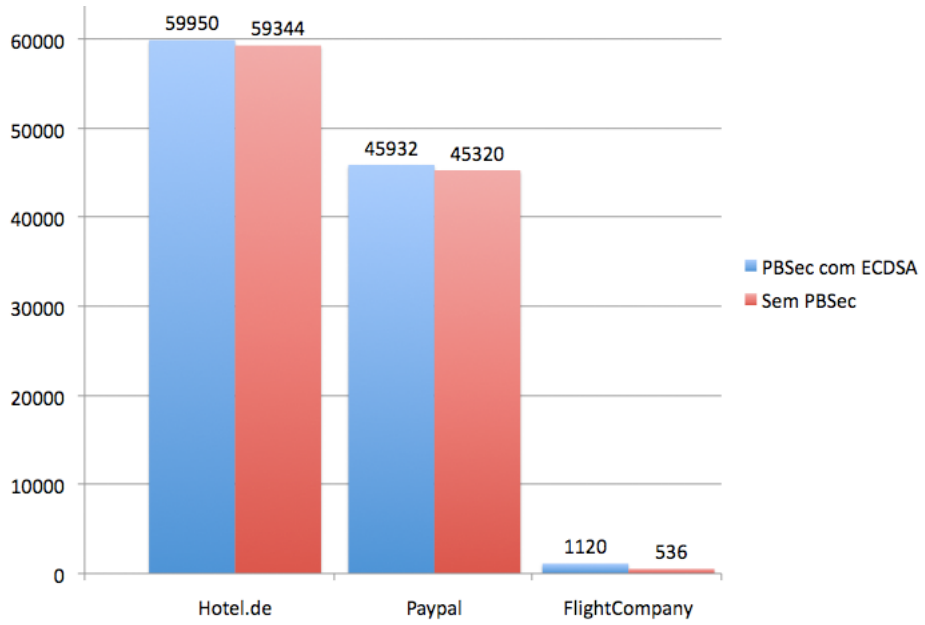

Figura 4.9: Resultado dos testes com criptografia ECDSA

Tabela 4.7: *Overhead* Com Criptografia ECDSA na Primeira parte da Conexão

Serviço	Tempo (ms)
Hotel.de	606
PayPal	612
FlightCompany	584

Com Criptografia DSA

A Figura 4.10, demonstra os resultados considerando o uso de criptografia DSA com chave de 1024 bits. Na Tabela 4.8 temos outros dados estatísticos retirados para dos testes, com criptografia DSA.

O *overhead* apresentado na Tabela 4.9 permite visualizar comparativamente com as Tabelas 4.5 e 4.7, que DSA é o algoritmos mais rápido dentre os algoritmos testados, tendo um aumento no tempo de comunicação de aproximadamente 4 vezes.

Tabela 4.8: Estatísticas dos resultados apresentados na Figura 4.10

Serviço	Valor	Desvio Padrão	Confiabilidade	Margem de Erro
Hotel.de	59797	21,00	95%	5
PayPal	45786	20,74	95%	5
FlightCompany	979	5,40	95%	5

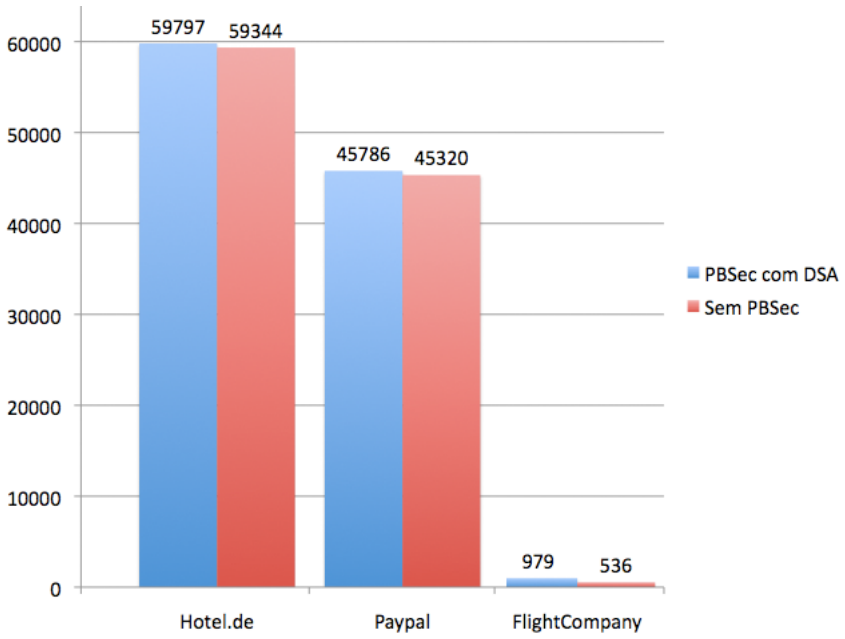

Figura 4.10: Resultado dos testes com criptografia DSA

Tabela 4.9: *Overhead* Com Criptografia DSA na Primeira parte da Conexão

Serviço	**Tempo (ms)**
Hotel.de	453
PayPal	466
FlightCompany	443

4.2.1 Considerações Finais

Com base no estudo de caso apresentado neste capítulo e de acordo com as situações propostas inicialmente nesta pesquisa, é possível perceber o impacto de um sistema intermediário no tratamento automático de políticas de segurança. Pode-se ainda perceber a diferença não só da inserção do PBSec na comunicação, mas também da variação entre os próprios algoritmos de criptografia.

Através da Tabela 4.10, podemos perceber que o *overhead* causado pelo PBSec varia entre 1 e 2 vezes o tempo de uma comunicação direta com o serviço, o que demonstra que a utilização de um sistema intermediário para estes casos não causa um *overhead* maior que duas vezes o tempo sem o sistema. Deve-se considerar que estes resultados foram obtidos com o sistema PBSec instalado na mesma rede local que o cliente, o que acaba por minimizar o *overhead* causado na primeira parte da comunicação. A situação representada pelo serviço *FlightCompany*, visa demonstrar o uso do PBSec como uma extensão do cliente, onde todos os serviços estão rodando em uma rede local, por este motivo os resultados divergem tanto dos apresentados pelos outros serviços.

Tabela 4.10: Comparativo entre os tempos de resposta encontrados no estudo.

Serviço	Original	Sem Criptografia	Vezes	RSA	Vezes	ECDSA	Vezes	DSA	Vezes
Hotel.de	59344	59454	**1,0018**	59922	**1,0097**	59950	**1,0102**	59797	**1,0076**
PayPal	45320	45440	**1,0026**	45905	**1,0129**	45932	**1,0135**	45786	**1,0102**
FlightCompany	536	651	**1,2145**	1103	**2,0578**	1120	**2,0895**	979	**1,8265**

A Tabela 4.11 apresenta um comparativo entre os *overheads* dos diferentes algoritmos de criptografia testados para o Serviço Hotel.de, demonstrando a quantidade de vezes que um é mais lento que o outro. O cálculo é realizado da seguinte forma: o item da linha X é dividido pelo item da coluna Y (e.g. RSA da linha X é 578 ms e o DSA da coluna Y é 453 ms, portanto X/Y ou 578/453 = 1,2759).

Tabela 4.11: Comparativo entre os *overheads* para o Serviços testados no estudo

Hotel.de				
y/x	**Sem Criptografia**	**RSA**	**ECDSA**	**DSA**
Sem Criptografia	-	5,2545	5,5091	4,1182
RSA	0,1903	-	1,0484	0,7837
ECDSA	0,1815	0,9538	-	0,7475
DSA	0,2428	1,2759	1,3377	-
Paypal				
y/x	**Sem Criptografia**	**RSA**	**ECDSA**	**DSA**
Sem Criptografia	-	4,8750	5,1000	3,8833
RSA	0,2051	-	1,0462	0,7966
ECDSA	0,1961	0,9559	-	0,7614
DSA	0,2575	1,2554	1,3133	-
FlightCompany				
y/x	**Sem Criptografia**	**RSA**	**ECDSA**	**DSA**
Sem Criptografia	-	4,9304	5,0783	3,8522
RSA	0,2028	-	1,0300	0,7813
ECDSA	0,1969	0,9709	-	0,7586
DSA	0,2596	1,2799	1,3183	-

Podemos visualizar nas Tabelas 4.10 e 4.11 que apesar do *overhead* total do PBSec não ser superior a duas vezes o tempo normal, a diferença entre ter criptografia e não ter criptografia pode causar um *overhead* até cinco vezes maior.

Antes de apresentarmos as conclusões encontradas durante a pesquisa, é importante fazer algumas considerações quanto aos testes realizados no estudo de caso. A implementação do estudo de caso apresenta limitações ao utilizar serviços reais, sendo que alguns dos serviços testados não funcionaram, devido a outras características de segurança não suportadas pelo protótipo do PBSec. O problema inicial foi encontrar serviços gratuitos com as características desejadas, os principais serviços encontrados são pagos. O serviço Web Services para pesquisa e compra de passagens aéreas da TAM, é pago; o serviço de cotações da bolsa é pago; alguns serviços de hotelaria pesquisados também eram pagos, o que acabou por inviabilizar a utilização destes. Por este motivo, que serviços de testes (estes

serviços são muito similares a suas versões pagas) foram adotados, como a API *SandBox* do PayPal e o serviço de passagens aéreas, desenvolvido em conjunto com o protótipo com o propósito de ser um serviço para testes iniciais.

Com base nos resultados apresentados ao longo desta seção, podemos começar a realizar um comparativo entre as hipóteses levantadas inicialmente na Introdução e os resultados encontrados, demonstrando a efetividade ou não da pesquisa e do modelo proposto, estes comparativos são apresentados no Capítulo 5, neste mesmo capítulo também são apresentadas possibilidades para a expansão deste trabalho.

5. CONCLUSÃO

Uma vez coletados os dados durante testes realizados com o estudo de caso e apresentados na Seção 4.2, puderam ser realizadas as respectivas análises, obtendo-se os resultados que permitiram chegar ao seguinte conjunto de conclusões.

O presente trabalho utilizou como guia as seguintes hipóteses:

- Sistemas baseados em SOA possuem diversos algoritmos de criptografia que dificultam o desenvolvimento de aplicações clientes, principalmente por diferentes empresas possuírem diferentes políticas de segurança.
- Pode-se facilitar o desenvolvimento de aplicações clientes, através da automatização do processo de segurança através de um intermediário auto-customizável.
- Um sistema intermediário pode ser utilizado para automatizar políticas de segurança possuindo um aumento aceitável no tempo de resposta do serviço.
- O algoritmo de criptografia possui uma importância significativa no tempo de resposta de um serviço criptografado.

Ao realizar uma avaliação considerando as hipóteses levantadas e os resultados obtidos, cada hipótese será abordada individualmente e os resultados serão utilizados para justificar as conclusões apresentadas.

Conforme o levantamento bibliográfico realizado e apresentado no Capítulo 2, é proposta da arquitetura SOA que as aplicações sejam compostas por serviços disponibilizados por diferentes empresas. Cada empresa é responsável por prover/requisitar segurança ao seu serviço, tendo em vista a existência de uma variedade muito grande de formas de se proporcionar segurança e que cada empresa pode adotar a sua sem restrições. Mesmo utilizando Web Services para desenvolver seus serviços, a tarefa de integrar serviços pode ser considerada relativamente complicada. A dificuldade na codificação e na manutenção de aplicações podem acabar determinando a profundidade da adoção da arquitetura ou modelo.

Este trabalho teve como um dos seus objetivos avaliar a hipótese que questiona se seria possível facilitar o desenvolvimento de aplicações clientes através de um serviço intermediário capaz de se adaptar automaticamente as características de segurança envolvidas,

especificamente algoritmos de criptografia assimétricos. Com base também nos trabalhos relacionados apresentados na Seção 2.4, outras pesquisas semelhantes apontam a possibilidade de auto-customização de características de serviços com base em sistemas intermediários. Desta forma, foi proposto um modelo e elaborado um protótipo, apresentados no Capítulo 3, que demonstram a efetividade deste tipo de abordagem.

Com a confirmação desta hipótese, pode-se partir para a seguinte, a qual trata sobre a viabilidade de se empregar um sistema intermediário para tratar das características de segurança em aplicações SOA. Com base nos resultados apresentados na Seção 4.2, principalmente na Tabela 4.10, podemos verificar que o *overhead* causado pelo PBSec é de no máximo o dobro do tempo normal de resposta, o que significa que, para aplicações onde a demanda não é muito grande e o tempo de resposta da aplicação é variavel, esta abordagem é válida.

Finalizando a análise das hipóteses, pode-se afirmar que, a partir da última hipótese, que trata da influência dos algoritmos de criptografia no tempo de resposta do serviço, podemos verificar na Seção 4.2, mais especificamente na Tabela 4.11, que existe uma variação entre os algoritmos de criptografia testados, mas que esta variância é sutil, algo entre 0,7 a 1,3 vezes mais rápidas que as outras. Nesta hipótese, podemos ressaltar que a maior diferença no tempo de *overhead* é entre serviços com e sem criptografia. Esta diferença pode aumentar o tempo de *overhead* de 3,8 a 5,5 vezes mais, apenas adicionando criptografia. Ainda podemos afirmar que, entre os algoritmos testados, o que teve o melhor desempenho na questão menor tempo de resposta para criptografia com uma chave de 1024 bits foi o DSA, com um tempo 1,0076 vezes maior que o tempo da comunicação direta entre o cliente e o serviço. Cabe dizer que este tempo somado ao desempenho normal é característico em todas as soluções que envolvem *proxys*, este custo pode aumentar conforme a demanda sobre o sistema.

Cabe ressaltar que as diferenças de *overhead* nos testes com PBSec e sem PBSec não são grandes devido ao fato de que o PBsec está na mesma rede dos clientes. Se estivessem em redes diferentes o impacto seria maior. Entretanto, o cliente pode instalar o PBSec na sua rede local para facilitar a implementação da sua aplicação, melhorando assim o tempo de resposta do sistema.

Além da análise das hipóteses é possível comparar o modelo proposto com outros modelos já existentes e abordados no Capítulo 2. Dentro do principal *framework* de segurança para Web Services há uma sub-especificação, o *WS-Trust*, que é responsável por garantir que os serviços possuam os requisitos de segurança solicitados pelo servidor. Comparativamente com o PBSec, eles não realizam exatamente a mesma função, o *WS-Trust* poderia ser parte do PBSec verificando as políticas de segurança solicitadas, porém ele não seria capaz de realizar a conversão entre diferentes políticas, função esta que viabiliza a comunicação entre serviços com diferentes pre-requisitos.

Outra comparação interessante que pode ser realizada, é com o artigo [JZ08] analisado na Seção 2.4. Este é um trabalho muito similar ao que foi proposto pela nossa dissertação. Porém, apesar de ser um *software* intermediário é necessário ter acesso ao cliente e ao servidor para que o serviço possa funcionar, visto que a sua implementação prevê a captura do fluxo de dados das placas de rede. Não existe o cadastro prévio dos serviços, sendo que desta forma é necessário estar nas duas pontas para efetuar as conversões. Estas limitações acabam inviabilizando o seu uso em ambientes reais, onde os serviços são providos por terceiros e não se tem acesso ao lado do servidor. Por estar em ambos os lados o modelo apresentado neste artigo, prima pela automatização do processo, visto que um cadastro não é necessário. Apesar de funcionar de maneira diferente do PBSec, a idéia básica por trás do modelo é a mesma, corroborando a sua usabilidade.

Com base nos resultados apresentados, podemos confirmar a efetividade do modelo. Demonstrando a sua viabilidade através de testes de desempenho, deve-se considerar que a força criptográfica dos diferentes algoritmos não foi considerada.

Ainda é possível efetuar melhorias no protótipo e estender o modelo a outros casos. Podemos citar como possíveis trabalhos futuros a extensão do modelo para tratamento de características de autenticação, como diversos *tokens* existentes. Pode-se ainda estender a quantidade de algoritmos de criptografia suportados, aumentando desta forma a abrangência do protótipo já existente.

Entre melhorias diretas no protótipo, o processo de cadastramento e geração dos serviços intermediários através do arquivo de descrição da interface do serviço pode ser melhorado, incluindo uma nova abordagem para tratar de exceções. Atualmente, as exceções devem ser contornadas manualmente, o que diminui a usabilidade do protótipo.

Tendo em vista ainda estender a análise com os dados já coletados, é possível considerar as forças criptográficas de diferentes algoritmos através da realização de uma comparação diferente entre estes. Para exemplificar esta situação, podemos considerar que supostamente um algoritmo DSA de 1024 bits possui a mesma força criptográfica que um ECDSA de 512 bits. Desta forma, poderia ser estabelecida uma comparação entre equivalência entre estes dois algoritmos, demonstrando realmente qual é o mais eficiente e rápido.

REFERÊNCIAS BIBLIOGRÁFICAS

[ACKM03] G. Alonso, F. Casati, H. Kuno, e V. Machiraju. *"Web Services: Concepts, Architectures and Applications"*. Springer, Berlin, Primeira Edição, Outubro, 2003, 354p.

[AGL+07] M. Ates, C. Gravier, J. Lardon, J. Fayolle, e B. Sauviac. "Interoperability Between Heterogeneous Federation Architectures: Illustration with SAML and WS-Federation". *In. 3rd International IEEE Conference on Signal-Image Technologies and Internet-Based System*, p. 1063–1070, 2007.

[AKD04] D. Aaker, V. Kumar, e G. Day. *"Pesquisa De Marketing"*. Editora Atlas, Segunda Edição, Fevereiro, 2004, 745p.

[Apa08] Apache. "Rampart". Capturado em: http://apache.rampart.org, Junho 2008.

[Apa09a] Apache. "Axis2". Capturado em: http://apache.axis2.org, Junho 2009.

[Apa09b] Apache. "Tomcat". Capturado em: http://tomcat.apache.org/, Julho 2009.

[BEK+00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, e D. Winer. "Simple Object Access Protocol 1.1". Technical Report, W3C, 2000.

[Bou09] BouncyCastle. "Bouncycastle". Capturado em: http://www.bouncycastle.org, Junho 2009.

[CB07] K. Canyang e D. Booth. "Web Services Description Language (WSDL) version 2.0". Technical Report, W3C, 2007.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, e S. Weerawarana. "Web Services Description Language". Technical Report, W3C, 2001.

[CHvRR04] L. Clement, A. Hately, C. von Riegen, e T. Rogers. "UDDI Technical Specification version 3.0.2". Technical Report, OASIS, 2004.

[CM05] A. Charfi e M. Mezini. “Middleware Services for Web Service Compositions”. *In.14th International Conference on World Wide Web: Special Interest Tracks and Posters*,p. 1132–1133, 2005.

[CZC08] M. Crasso, A. Zunino, e M. Campo. “Easy Web Service Discovery: A Query-by-Example Approach”. *Science of Computer Programming*, vol. 71, Abril, 2008, p. 144–164.

[EMT06] A. Erradi, P. Maheshwari, e V. Tosic. “Policy-Driven Middleware for Self-Adaptation of Web Services compositions”. *In. Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, p. 62–80, 2006.

[Erl05] T. Erl. *“Service-Oriented Architecture: Concepts, Technology, and Design”*. Prentice Hall PTR, Primeira Edição, Agosto, 2005, p.792.

[FH08] C. Farkas e M. Huhns. “Securing Enterprise Applications: Service-Oriented Security (SOS)”. *In. 10th IEEE Conference on E-Commerce Technology and 5th IEEE Conference on Enterprise Computing, E-Commerce and E-Services*, p. 428–431, 2008.

[FKK96] A. Freier, P. Karlton, e P. Kocher. “The SSL Protocol version 3.0”. Technical Report, Transport Layer Security Working Group, 1996.

[GKM+07] M. Gudgin, A. Karmarkar, J. Moreau, Y. Lafon, N. Mendelsohn, H. Frystyk, e M. Hadley. “SOAP version 1.2”. Technical Report, W3C, 2007.

[HBN+04] H. Haas, D. Booth, E. Newcomer, M. Champion, D. Orchard, C. Ferris, e F. McCabe. “Web Services Architecture”. Technical Report, W3C, 2004.

[HYO+07] M. Hondo, P. Yendluri, P. Orchard, F. Hirsch, A. Vedamuthu, T. Boubez, e U. Yalçinalp. “Web Services Policy 1.5 - Framework”. Technical Report, W3C, 2007.

[HYS06] S. Haitao, Y. Yingyu, e Y. Shixiong. “Dynamic Aspects Weaving in Service Composition”. *In. 6th International Conference on Intelligent Systems Design and Applications*, p. 1003–1008, 2006.

[IR09] L. Iacono e H. Rajasekaran. “Secure Browser-Based Access to Web Services”. *In. IEEE International Conference on Communications 2009*, p. 1–5, 2009.

[JE07] D. Jordan e J. Evdemon. “Web Services Business Process Execution Language version 2.0”. Technical Report, OASIS, 2007.

[JZ08] W. Jian e H. Zhimin. “Proxy-based web service security”. *In. Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference*, p. 1282–1288, 2008.

[KM06] C. Kaler e M. McIntosh. “Web Services Federation Language (WS-Federation)”. Technical Report, OASIS, 2006.

[Lab99] RSA Laboratories. “Pkcs #12: Personal Information Exchenge Syntax Standard”. Technical Report, RSA Laboratories, 1999.

[Mic09] Microsoft. “Dcom Technical Overview”. Capturado em: http://technet.microsoft.com, Agosto 2009.

[MLM+06] M. MacKenzie, K. Laskey, F. McCabe, P. Brown, e R. Metz. “Reference Model for Service Oriented Architecture 1.0”. Technical Report, OASIS, 2006.

[MMNS06] K. Mostefaoui, Z. Maamar, N. Narendra, e S. Sattanathan. “Decoupling Security Concerns in Web Services Using Aspects”. *In. 3rd International Conference on Information Technology: New Generations*, p. 20–27, 2006.

[New02] E. Newcomer. *“Understanding Web Services: XML, WSDL, SOAP, and UDDI”*. Addison-Wesley Professional, Primeira Edição, Maio, 2002, 368p.

[NGG+07a] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, e H. Granqvist. “WS-SecureConversation 1.3”. Technical Report, OASIS, 2007.

[NGG+07b] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, e H. Granqvist. “WS-Trust 1.3”. Technical Report, OASIS, 2007.

[NIS99] NIST. “FIPS pub 46-3: Data Encryption Standard (DES)”. Technical Report, National Institute of Standards and Technology, 1999.

[NIS01] NIST. “FIPS pub 197: Advanced Encryption Standard (AES)”. Technical Report, National institute of standards and technology, 2001.

[oEE00] Institute of Electrical and Electronics Engineers. “IEEE Standard Specifications for Public-Key Cryptography”. Technical Report, IEEE p.1363, 2000.

[O’N03] M. O’Neill. *“Web Services Security”*. McGraw-Hill Osborne Media, Primeira Edição, Janeiro, 2003, 312p.

[oST93] National Institute of Standards and Technology. “Electronic data interchange (EDI)”. Technical Report, FIPS 161-2, 1993.

[oST94] National Institute of Standards and Technology. “Digital Signature Standard”. Technical Report, FIPS 186, 1994.

[Pay09] PayPal. “Paypal Sandbox”. Capturado em: https://developer.paypal.com/, Julho 2009.

[PMSM+08] J. Paoli, E. Maler, C. Sperberg-McQueen, Bray T., e F. Yergeau. “Extensible Markup Language (XML) 1.0”. Technical Report, W3C, 2008.

[Pos82] P. Postel. “Simple Mail Transfer Protocol”. Technical Report, RFC 821, 1982.

[Ros01] M. Rose. “The Blocks Extensible Protocol Core”. Technical Report, RFC 3080, 2001.

[RR07] L. Richardson e S. Ruby. *“RESTful Web Services”*. O’Reilly Media, Inc., Primeira Edição, Maio, 2007, 448p.

[RS99] E. Rescorla e A. Schiffman. “The Secure Hypertext Transfer Protocol”. Technical Report, RFC 2660, 1999.

[RSA78] R. Rivest, A. Shamir, e L. Adleman. “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”. *Communications of the ACM*, vol. 21-2, Fevereiro, 1978, p. 120–126.

[Sch94] B. Schneier. “Description of a New Variable-length Key, 64-bit Block Cipher (Blowfish)”. *In. Cambridge Security Workshop on Fast Software Encryption*, p. 191–204, 1994.

[Sin07] A. Singhal. “Web Services Security: Challenges and Techniques”. *In. 8th IEEE International Workshop on Policies for Distributed Systems and Networks*, p. 282, 2007.

[SLLCJW07] S. Shun-Lung, W. Lih-Chyau, e J. Jhih-Wei. “A New 256-bits Block Cipher - Twofish256”. *In. International Conference on Computer Engineering & Systems*, p. 166–171, 2007.

[Spe08] Object Management Group Specification. “Documents Associated with Corba, 3.1”. Technical Report, OMG, 2008.

[Sri95] R. Srinivasan. “RPC: Remote Procedure Call Protocol Specification version 2”. Technical Report, RFC 1831, 1995.

[SSV07] S Souza, C. Schepke, e V. Viana. “Avaliação de Desempenho de SOAP sobre HTTP, SMTP e BEEP”. *In. Anais da V Escola Regional de Redes de Computadores, ERRC*, Disponível em http://www.inf.ufrgs.br/~cschepke/outras/errc07.pdf, 2007.

[STK01] J. Snell, D. Tidwell, e P. Kulchenko. *“Programming Web Services with SOAP”*. O’Reilly Media, Inc., Primeira Edição, Dezembro, 2001 264p.

[Sun09] Sun. “Java Cryptography Extension - Reference Guide”. Capturado em: http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html, Junho 2009.

[Tan02] A. Tanenbaum. *“Computer Networks”*. Prentice Hall PTR, quarta Edição, Agosto, 2002, 945p.

[Ver09] Verisign. “Certificadora Verisign”. Capturado em: http://www.verisign.com.br, Julho 2009.

[WRW96] A Wollrath, R. Riggs, e J. Waldo. “A Distributed Object Model for the Java System”. *In. USENIX Computing Systems*, p. 265–290, 1996.

[Yer96] F. Yergeau. “UTF-8, a Transformation Format of Unicode and ISO 10646”. Technical Report, RFC 2044, 1996.

[ZPR03] A. Zorzo, P. Periorellis, e A. Romanovsky. “Using Co-ordinated Atomic Actions for Building Complex Web Applications: a Learning Experience”. *In. Proceedings of the 8th International Workshop on Object-Oriented Real-Time Dependable Systems*, p. 288–295, 2003.

APÊNDICE A - Configuração Serviços

As Figuras A.1, A.2 e A.3 apresentam o funcionamento do processo de compilação e de geração do serviço, que no caso do *framework* Axis, é um arquivo com a extensão aar, comparável com um arquivo de pacote jar, tendo em vista que contém todas as classes java e chaves criptográficas necessárias. Na Figura A.1, são definidas as variáveis de ambiente e os locais de armazenamento. A Figura A.2 , mostra o processo de compilação, onde são definidos os parâmetros para o "*javac*" e direcionados os arquivos compilados. Já na Figura A.3 os arquivos compilados são agrupados em um pacote que forma o serviço, no caso do Axis, um serviço com extensão aar.

O arquivo aar é composto de uma pasta com o nome do serviço que contêm os arquivos .class compilados, uma pasta "META-INF" que contêm um arquivo "services.xml"com descrições sobre o fluxo de dados e forma de criptografia. Ainda temos um arquivo com o *Keystore*, que contêm as chaves criptográficas necessárias e um arquivo .properties, que possui definições das propriedades do *Keystore*.

Conforme mencionado anteriormente, as chaves ficam armazenadas no formato PKCS12, na Figura A.4, com o auxílio de uma interface gráfica (*Keytool IUI*) para a ferramenta *Keytool*, podemos visualizar como as chaves são armazenadas dentro do *Keystore*.

```
<project basedir="." default="help">

    <property environment="env"/>
    <property name="AXIS2_HOME" value="${env.AXIS2_HOME}"/>

    <property name="srcDir"             value="src"/>
    <property name="buildDir"           value="build"/>
    <property name="classDir"           value="${buildDir}/classes"/>
    <property name="pkgFib"             value="fibonacci"/>
    <property name="aarFile"            value="${buildDir}/FibonacciService.aar"/>
    <property name="aarFilePrekey"      value="${buildDir}/FibonacciServicePrekey.aar"/>

    <path id="axis2.classpath">
        <fileset dir="${AXIS2_HOME}/lib">
            <include name="*.jar"/>
        </fileset>
        <pathelement path="${classDir}" />
        <pathelement path="bcprov-jdk15-141.jar" />
        <pathelement path="." />
    </path>

```

Figura A.1: Definição de variáveis no Apache Ant.

```
    <!-- ================================================================== -->
    <!-- Compile Java sources                                               -->
    <!-- ================================================================== -->

    <target name="compile">
        <mkdir dir="${buildDir}"/>
        <mkdir dir="${classDir}"/>

        <javac debug="on"
               fork="true"
               destdir="${classDir}"
               srcdir="${srcDir}"
               classpathref="axis2.classpath">
        </javac>
    </target>

    <target name="clean">
        <delete dir="${buildDir}"/>
        <delete file="${aarFile}"/>
    </target>

```

Figura A.2: Processo de Compilação

```
43      <!-- ================================================================== -->
44      <!-- Create service aar file                                            -->
45      <!-- ================================================================== -->
46
47      <target name="generate.service" depends="compile">
48          <!--aar them up -->
49
50          <!-- public key encryption -->
51          <delete>
52              <fileset dir="${classDir}" excludes="**/*.class"/>
53          </delete>
54          <copy toDir="${classDir}" failonerror="false">
55              <fileset dir="${basedir}/resources">
56                  <include name="**/*.*"/>
57              </fileset>
58          </copy>
59          <jar destfile="${aarFile}">
60              <fileset excludes="**/*Client.class" dir="${classDir}"/>
61          </jar>
62      </target>
63
```

Figura A.3: Processo de geração do serviço

Private Key (keypair) & Trusted Certificate Entries:

Alias	Entry	Valid Date ?	Self-Signed ?	Trusted C.A. ?	Key Size	Cert. Type	Cert. Sig. Algo.	Modified Date
client			✓	-	1024 bits	X.509	MD5withRSA	17/09/2008
service			✓	-	1024 bits	X.509	MD5withRSA	17/09/2008

Figura A.4: Certificados e chaves armazenados em uma *keystore*