UNIFACS UNIVERSIDADE SALVADOR
MESTRADO ACADÊMICO EM SISTEMAS E COMPUTAÇÃO

ALLAN DELON BARBOSA ARAÚJO

PERSEC: UM MIDDLEWARE PARA MULTICRIPTOGRAFIA EM WEB SERVICES (versão SDN)

Salvador
2015

ALLAN DELON BARBOSA ARAÚJO

PERSEC: UM MIDDLEWARE PARA MULTICRIPTOGRAFIA EM WEB SERVICES (versão SDN)

Dissertação apresentada ao Curso de Mestrado Acadêmico em Sistemas e Computação, Universidade Salvador UNIFACS, Laureate International Universities, como requisito parcial para a obtenção do título de Mestre.

Orientador: Prof. Dr. Paulo Caetano da Silva.

Salvador
2015

FICHA CATALOGRÁFICA
Elaborada pelo Sistema de Bibliotecas da UNIFACS Universidade Salvador, Laureate International Universities)

Araújo, Allan Delon Barbosa

PERSEC: um middleware para multicriptografia em web services (versão SDN)./ Allan Delon Barbosa Araújo.- Salvador: UNIFACS, 2015.

117 f. : il.

Dissertação Programa de Pós-Graduação em Sistemas e Computação de UNIFACS Universidade Salvador, Laureate International Universities como requisito parcial à obtenção do título de Mestre.

Orientador: Prof. Dr. Paulo Caetano da Silva.

1. Multicriptografia. 2. Redes de computação -- Medidas de segurança. I. Silva, Paulo Caetano da, orient. II. Título.

CDD: 005.82

ALLAN DELON BARBOSA ARAÚJO

PERSEC: UM *MIDDLEWARE* PARA MULTICRIPTOGRAFIA EM *WEB SERVICES*
(versão SDN)

Dissertação de Mestrado aprovada como requisito parcial para obtenção do grau de mestre em Sistemas e Computação, UNIFACS Universidade Salvador, Laureate International Universities pela seguinte banca examinadora:

Paulo Caetano da Silva – Orientador _____
Ph.D. in Computer Science, Universidade Federal de Pernambuco - UFPE
UNIFACS Universidade Salvador, Laureate International Universities

Roque Mendes Prado Trindade _____
Doutor em Engenharia Elétrica e de Computação pela Universidade Federal do Rio Grande do Norte
Universidade Estadual do Sudoeste da Bahia - UESB

Ernesto Massa _____
Doutor em Ciência da Computação - Ufba - Unifacs
UNIFACS Universidade Salvador, Laureate International Universities

Salvador, 31 de outubro de 2015.

AGRADECIMENTOS

Ao final desta longa jornada, que representou este Mestrado para mim, cabe agradecer, de forma particular e especial, a algumas pessoas, sem as quais o percurso não seria concluído. Assim, agradeço, de coração:

- A *Deus*, pelo dom da vida, por todas as dádivas a mim concedidas através de Sua infinita bondade e generosidade, pelas adversidades enfrentadas e pelas batalhas vividas (sendo este Mestrado uma das mais árduas), pois as encaro como oportunidades de crescimento e de amadurecimento como ser humano, portanto, muito obrigado, Senhor, por me agraciar com mais esta conquista;
- À minha mãe, *Maria José Barbosa Araújo*, por todo cuidado e atenção que tem me dispensado ao longo da minha vida. Isso me deu ânimo de sobra para buscar meus objetivos e, assim, realizar meus sonhos;
- Ao meu pai, *Joselito Machado Araújo*, por todo o suporte que me propiciou em vida como pai e, principalmente, por ter me convencido a continuar estudando quando, em uma determinada etapa, eu pensei em abandonar os estudos para começar a trabalhar (por isso, esta dissertação de Mestrado é um prêmio que dedico a ele, com carinho);
- Aos meus colegas de trabalho do IFBAIANO, *campus* Santa Inês, especialmente aos professores do curso de Informática (do qual também sou professor): *Cláudia Bochesse* e *Marcos Pereira*, pelo apoio dado ao me substituírem em sala de aula, quando necessitei me ausentar por conta das atividades do Mestrado (sem esse apoio, confesso que a tarefa de concluir com sucesso este curso seria muito mais complicada, por isso, registro aqui meus agradecimentos sinceros a estes verdadeiros colegas);
- À minha namorada, amiga e colega de trabalho, *Sílvia Pereira dos Santos*, pelo companheirismo, apoio moral e incentivo que me deu durante toda a trajetória do Mestrado, compartilhando comigo as expectativas, desafios e vitórias que ela mesma experimentava no seu curso de Mestrado (essa cumplicidade foi de fundamental importância, pois serviu de combustível para mim nos momentos mais cruciais);
- Ao professor de Informática do Instituto Federal da Bahia (IFBA), *Alexandro Santos Silva*, pela parceria no desenvolvimento do *middleware* proposto neste trabalho (a sua habilidade em programação *java* foi fundamental para a concretização deste projeto).
- Ao meu orientador, Paulo Caetano da Silva, pela excelência do seu trabalho em me orientar, apontando caminhos, analisando meticulosamente todo o trabalho realizado, sempre de forma paciente e amigável.

A todos vocês, mais uma vez, muito obrigado!

RESUMO

Os *Web Services* representam uma maneira de compartilhar dados e devem ser tratados como uma solução para a interoperabilidade entre sistemas heterogêneos. No entanto, por terem uma infraestrutura pública, sujeitos a ataques, problemas de segurança, tornaram-se desafiadores. Para garantir o uso seguro dessas aplicações, algumas especificações de segurança são utilizadas para promover a confidencialidade e integridade das informações, tais como: a XML *Signature* e XML *Encryption*. Não obstante, o processo de criptografia realizado por meio da especificação XML *Encryption* degrada o desempenho dos *Web Services*. Apesar disso, essa especificação de segurança XML tem sido amplamente utilizada para garantir a segurança ao nível de mensagem (segurança fim a fim) em aplicações *Web Services*. Isso se justifica porque, se esse tipo de segurança não for estabelecido, as mensagens SOAP, compartilhadas entre a aplicação cliente e o *Web Service*, ficariam expostas entre um ponto e outro da comunicação em meio às aplicações, revelando, indevidamente, dados de caráter confidencial. Considerando-se esse panorama, foi realizada neste trabalho uma revisão bibliográfica a fim de se detectarem evidências científicas que norteassem a proposta de uma nova solução de segurança para *Web Services*, a nível de mensagem, objetivando reduzir o impacto degradativo ocasionado pelo processo de criptografia. Neste sentido, a principal evidência científica detectada para tratar o problema em questão consistiu na hipótese de serem utilizados diversos algoritmos criptográficos para criptografar os dados de uma mensagem SOAP. Tal hipótese foi implementada por meio de um *middleware*, cuja base é um XML *Schema* para estruturar os dados de uma mensagem SOAP em níveis de confidencialidade. Dessa maneira, o *middleware* criptografaria cada nível por meio de um algoritmo criptográfico específico – levando em consideração a robustez destes algoritmos criptográficos em termos de segurança, ou seja, os níveis de mais baixa confidencialidade são criptografados por meio de algoritmos menos seguros, enquanto os níveis de mais alta confidencialidade são criptografados por meio de algoritmos mais seguros (como os algoritmos criptográficos possuem custos computacionais distintos, ponderou-se que esta técnica resultaria em uma melhor performance em comparação com o procedimento usual de se aplicar apenas um algoritmo criptográfico que, eventualmente, poderia ser o mais custoso computacionalmente para se realizar essa tarefa). Assim, com o intuito de se comprovar a efetividade de tal técnica, propôs-se uma avaliação experimental, com base nos principais cenários de uso do *middleware* (que permitem criptografar os dados de uma mensagem SOAP por meio de um, dois ou três algoritmos criptográficos), para averiguar se, de fato, a possibilidade de se aplicar diversos algoritmos criptográficos, para se criptografar uma mensagem SOAP aprimoraria o desempenho dos *Web Services* que implementam a segurança fim a fim. Os resultados obtidos com a execução dos experimentos comprovaram que o processo da criptografia, como também da decriptação, dos experimentos que utilizaram dois ou três algoritmos criptográficos para codificar uma determinada mensagem SOAP tiveram seu tempo de processamento significativamente reduzido em relação aos experimentos que empregaram apenas um algoritmo criptográfico para a mesma finalidade. Tal constatação serviu para validar a hipótese sustentada neste trabalho.

Palavras-chave: SOAP. *Web Services*. Especificação de segurança XML *Encryption*. Performance. XML *Schema*. Algoritmos criptográficos.

ABSTRACT

Web services are a way to share data and should be treated as a solution to interoperability between heterogeneous systems. However, by having a public infrastructure subject to attacks, security concerns have become challenging. To ensure the security of these applications, some security specifications are used to promote the confidentiality and integrity of information, such as the XML Signature and XML Encryption. However, the encryption process performed by the XML encryption specification degrades the performance of Web Services. Nevertheless, this XML security specification has been widely used to ensure the safety message-level (safety end-to-end), Web Services applications. This is justified because if this type of security is not established, SOAP messages, shared between the client application and the Web service, would be exposed from one point to another of communication between applications, revealing unduly confidential data. Considering this background, there was this work a literature review in order to detect scientific evidence that guide the proposal for a new security solution for Web services, message-level, aiming to reduce the degradation impact caused by the encryption process. Accordingly, the main scientific evidence detected to treat the problem in question was the suggestion to use different cryptographic algorithms for encrypting the data of a SOAP message. This hypothesis has been implemented by means of a middleware, based on an XML Schema for structuring data in a SOAP message confidentiality levels so that, in this way, the middleware was able to encrypt each level by a specific cryptographic algorithm - taking into account the robustness of these cryptographic algorithms in terms of safety, i.e., levels lower confidentiality are encrypted using less secure algorithms, while the highest levels of confidentiality are encrypted using safer algorithms (such as the cryptographic algorithms have fees separate computer, if reasoned that this technique would result in better performance in comparison with the usual procedure to apply only a cryptographic algorithm that eventually could be the most expensive computationally to accomplish this task). Therefore, in order to prove the effectiveness of this technique has been proposed an experimental evaluation, based on the main middleware usage scenarios (which allow encrypting the data of a SOAP message via one, two or three cryptographic algorithms), to see if, in fact, the possibility of applying different cryptographic algorithms, to encrypt a SOAP message, would improve the performance of Web Services that implement the security end-to-end. The results obtained with the execution of the experiments showed that the encryption process, as well as the decryption, the experiments using two or three cryptographic algorithms to encrypt a particular SOAP message, had its processing time significantly reduced from experiments only a cryptographic algorithm employed for the same purpose. This finding served to validate the hypothesis supported this work.

Keywords: SOAP. Web Services. Encryption XML security specification. Performance. Schema XML. Cryptographic algorithms.

LISTAS DE FIGURAS

LISTAS DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Este Capítulo apresenta a contextualização sobre o cenário do problema abordado nesta Dissertação, bem como a motivação, as justificativas, os objetivos, a metodologia de pesquisa e, por fim, a organização dos demais capítulos que compõem este trabalho.

1.1 CONTEXTUALIZAÇÃO DO PROBLEMA

Para se discutir a respeito dos *Web Services*, é preciso, antes, descrever a tecnologia sobre a qual os mesmos estão estruturados: SOA (*Service-Oriented Architecture*). Segundo Erl (2005), SOA pode ser definido como uma espécie de arquitetura de *software* baseada em processos de negócio, distribuindo funções como serviços, fornecendo uma série de princípios que servem como diretrizes para as fases de desenvolvimento e integração de sistemas, o que resulta em uma série de funcionalidades que são disponibilizadas como serviços interoperáveis, por meio de uma interface que viabiliza a comunicação entre clientes e serviços.

Resumidamente, as características essenciais da tecnologia SOA podem ser assim descritas:

a) **Serviço** – compreendido como uma função computacional, é disponibilizado por meio de uma interface bem definida para os clientes interessados;
b) **Sem Estado (*Stateless*) –** significa a independência de um serviço em relação a outro, exceto no que diz respeito aos serviços coordenados;
c) **Descoberta** – diz respeito à capacidade de se efetuar a localização e identificação dos serviços por meio de um repositório central;
d) **Coordenação** – característica compreendida como a capacidade de organizar os serviços sequencialmente, objetivando realizar uma atividade específica;
e) ***Binding*** – representa, basicamente, a conexão dinâmica entre o cliente e o serviço.

De maneira geral, atualmente, a tecnologia *Web Services* é o meio mais difundido para uma implementação SOA.

Os *Web Services* são considerados como a evolução natural dos modelos de computação distribuída difundidos na década de 90, como o RMI (*Remote Method Invocation*), o DCOM (*Distributed Component Object Model*) e o CORBA (*Common Object Request Broker Architecture*). Essas tecnologias vinham sendo utilizadas apenas na integração de *softwares* em ambientes de redes locais e homogêneos. Entretanto, a partir do uso da

Internet no mercado corporativo, as tecnologias retrocitadas se revelaram inadequadas devido à heterogeneidade envolvida na integração dos diversos ambientes tecnológicos oriundos de cada corporação.

Diante de tal cenário, um consórcio entre empresas, denominado W3C (*World Wide Web Consortium*), composto por grandes empresas de TI (Tecnologias da Informação) – como a IBM, Microsoft e BEA – por meio da junção das tecnologias empregadas para o desenvolvimento Web, dentre elas, JSP (*JavaServer Pages*), ASP (*Active Server Pages*), PHP (*Hypertext Preprocessor*) e XML (*EXtensible Markup Language*), criou o padrão SOAP (*Simple Object Accse Protocol*) para o desenvolvimento de aplicações capazes de se integrar em ambientes heterogêneos, ou seja, os *Web Services* (WS) – uma tecnologia que propicia a integração de aplicações, enviando ou recebendo informações, independentemente da linguagem de programação em que foram desenvolvidas, do sistema operacional nos quais são executadas e do *hardware* em que são utilizadas (GOMES, 2009).

Segundo a definição do *World Wide Web Consortium* (W3C), um *Web Service* é um componente de *software* identificado por um URL (*Uniform Resource Locator*), cujas interfaces e ligações são capazes de ser definidas, descritas e descobertas como artefatos XML. Um *Web Service* suporta interações diretas com outros componentes de *software* usando mensagens baseadas em XML, trocadas via protocolos na *Internet* (W3C, 2004).

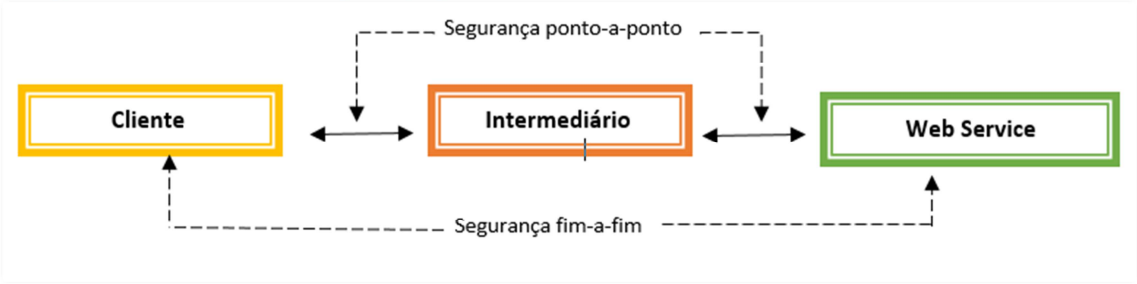
A interoperabilidade entre sistemas heterogêneos, considerando tanto componentes de *hardware* como de *software*, promovida pelos *Web Services*, acontece em razão de se utilizar padrões abertos baseados em XML (*EXtensible Markup Language*), tais como: SOAP (*Simple Object Access Protocol*) – usado para a transferência de dados entre os serviços; WSDL (*Web Services Description Language*) – define um esquema XML para descrever os serviços disponíveis; e o UDDI (*Universal Description, Discovery and Integration*) – determina um modo de publicar e descobrir informações sobre um serviço específico em um diretório ou registro de serviços UDDI (JOSUTTIS, 2007). Além disso, a ubiquidade dos protocolos da *Internet* e o formato de dados, tais como o *Hypertext Transfer Protocol* (HTTP) e o *Extensible Markup Language* (XML), permitem transferir informações de forma transparente através de *firewalls* e servidores *proxy*, tornando os *Web Services* componentes facilmente acessíveis por parte de qualquer aplicação (W3C, 2004).

Entretanto, apesar dos potenciais benefícios alcançados pelos *Web Services* como uma tecnologia de sistemas distribuídos, não é possível se destacar a confidencialidade das informações transitadas – as quais, muitas vezes, possuem caráter confidencial – sem a utilização de técnicas apropriadas para essa finalidade. Assim, garantir a segurança das

informações é uma questão de vital importância ao se utilizar *Web Services*, uma vez que essas aplicações estão sujeitas a ataques que podem expor seus processos e fluxos de negócio, levando a consequências financeiras ou legais graves, caso tais informações sejam interceptadas por terceiros ou, até mesmo, fraudadas e aceitas como válidas (SOSNOSKI, 2009).

Nesse contexto, garantir a segurança das informações do usuário dentro de uma mensagem SOAP representa um grande desafio (O'NEILL, 2003), considerando o fato de que essa mensagem pode trafegar entre diversos *Web Services* para atender a requisições de um cliente. Tal desafio consiste em assegurar que as informações contidas na mensagem sejam acessíveis apenas pelo *Web Service* requisitado e pelo cliente envolvido na requisição, ou seja, segurança fim a fim. Dessa forma, tecnologias como o SSL/TLS (*Secure Sockets Layer/Transport Layer Security*) têm como objetivo garantir a segurança ponto a ponto, mas, por outro lado, não garantem a segurança fim a fim, isto é, a segurança em nível de mensagem. Entretanto, esse tipo de segurança é necessário em um ambiente de *Web Services*, devido ao possível trânsito das mensagens SOAP por *Web Services* intermediários até alcançar o destinatário final. Logo, com a utilização de criptografia apenas na camada de transporte, por meio das tecnologias SSL/TLS, corre-se o risco de que as informações sejam reveladas aos *Web Services* intermediários entre o emissor e o receptor da mensagem SOAP, devido a lacunas de segurança existentes entre os *Web Services* intermediários localizados no percurso da mensagem SOAP (MASHOOD; WIKRAMANAYAKE, 2007). A Figura 1 ilustra esses contextos de segurança discutidos para *Web Services*.

Figura 1 – Contexto de Segurança para Web Services



Fonte: Adaptado de Microsoft (2002).

O protocolo SOAP, por sua vez, não define nenhum mecanismo de segurança, valendo-se apenas de mecanismos em nível de infraestrutura de redes que, geralmente, tratam apenas da segurança no nível da camada de transporte. Dessa forma, novas soluções e novos padrões de segurança têm sido propostos para tratar da segurança fim a fim, destacando-se,

entre elas, as especificações de segurança XML *Encryption* e a XML *Signature* (SILVA; CUNHA, 2005).

Entretanto, a aplicação de tais especificações pode degradar, consideravelmente, o desempenho dos *Web Services*, pois resultam na adição de novos elementos XML à mensagem original, ocasionando, consequentemente, (i) – um maior consumo da largura de banda de rede para garantir o tráfego das mensagens SOAP que fazem uso dessas especificações e (ii) – um maior consumo dos recursos computacionais da CPU para o devido processamento dessas mesmas mensagens (LIU et al., 2005) (ENGELEN; ZHANG, 2008b) (GRUSCHKA et al., 2011).

Além do mais, vale ressaltar o papel relevante da especificação XML *Encryption* na referida degradação de desempenho. Apesar de essa especificação possibilitar a criptografia apenas dos elementos da mensagem que realmente necessitam de segurança, o que leva a uma economia computacional importante dos recursos de processamento da CPU, considerando o alto custo computacional da operação de criptografia (NAGAPPAN et al., 2003), os algoritmos criptográficos envolvidos, bem como o tamanho das chaves públicas utilizadas, determinam alto impacto degradativo no desempenho dos *Web Services* que fazem uso dessa especificação de segurança XML (RODRIGUES; BRANCO, 2009).

Neste sentido, observa-se que, geralmente, as soluções de segurança, que visam garantir a segurança fim a fim às mensagens SOAP, utilizam apenas um algoritmo criptográfico em conjunto com uma chave simétrica para a criptografia dos dados que compõem a mensagem SOAP. Este trabalho, por sua vez, se baseia na hipótese, sugestionada por Rodrigues e Branco (2009), de que a aplicação combinada de diversos algoritmos para a criptografia de uma mensagem SOAP, talvez, resulte em uma melhora de desempenho, se comparada com o procedimento convencional de se aplicar apenas um algoritmo para essa tarefa, principalmente quando a criptografia precisa ser realizada em uma ampla gama de dados de uma mensagem SOAP, empregando, para tanto, um algoritmo criptográfico de alto custo computacional.

Considerando o cenário descrito, diversos estudos têm sido desenvolvidos com o objetivo de sanar ou, pelo menos, amenizar o impacto do custo computacional agregado à execução dos *Web Services* que implementam a segurança fim a fim às mensagens SOAP (geralmente, por meio das especificações de segurança XML). O presente trabalho soma-se a essa categoria de estudos, propondo uma solução cujo objetivo seja não somente garantir a segurança da informação transitada na *Internet*, como também a otimização do desempenho

dos sistemas, abordando, em especial, o problema de degradação dos *Web Services* causado pelo processo de criptografia dos dados.

1.2 JUSTIFICATIVAS

A partir da contextualização do problema, realizada na Seção 1.1, as justificativas para o desenvolvimento deste estudo consistem em dois fatores:

a) promover a segurança fim a fim das mensagens compartilhadas entre os *Web Services* – o que representa uma questão de fundamental importância, uma vez que a exposição dos mesmos a ataques poderia comprometer o fluxo de negócio dessas aplicações, acarretando graves problemas financeiros ou legais se, por acaso, as informações transitadas entre os *Web Services* e as aplicações clientes fossem capturadas ou, até mesmo, corrompidas por terceiros, sendo posteriormente aceitas como válidas pelas partes legítimas envolvidas na comunicação;
b) promover uma melhora de desempenho dos *Web Services* que produza a segurança fim a fim, por meio da técnica de multicriptografia de mensagens SOAP (proposta neste trabalho), visto que a prática usual de se aplicar apenas um algoritmo criptográfico, considerando seu custo computacional, para criptografar todos os dados confidenciais de uma mensagem SOAP de maneira uniforme, tem gerado um impacto degradativo considerável no desempenho desses sistemas (RODRIGUES; BRANCO, 2009).

1.3 MOTIVAÇÃO

Como foi descrito na Seção 1.1, o processo de criptografia XML em *Web Services* implica problemas de perda de desempenho nessas aplicações. Este trabalho busca tratar de tal problema, discutindo não apenas quais são as partes de uma mensagem SOAP que devem ser necessariamente criptografadas, mas também como essas partes devem ser criptografadas, levando-se em consideração a robustez e também o custo computacional dos algoritmos criptográficos que são, comumente, empregados para a criptografia dos dados XML. Observa-se que, geralmente, as soluções de segurança, que visam garantir a segurança fim a fim às mensagens SOAP, utilizam somente um algoritmo criptográfico em conjunto com uma chave simétrica para a criptografia dos dados que compõem essas mensagens.

Mas, ao se analisar o trabalho de Rodrigues e Branco (2009), no qual os autores realizam um estudo comparativo sobre as especificações de segurança aplicadas à arquitetura de serviços (SOA) e sobre a eficácia dos algoritmos criptográficos mais comumente utilizados, detectaram-se duas situações:

a) o uso da criptografia é o que causa maior impacto no desempenho das aplicações baseadas em *Web Services*;
b) alguns algoritmos criptográficos, como o 3DES, são os mais custosos em termos de processamento do que outros.

Com base nesses dois fatores, Rodrigues e Branco (2009) consideram uma hipótese: a aplicação combinada de diversos algoritmos para a criptografia de uma mensagem SOAP pode resultar em uma melhora de desempenho, se comparada com o procedimento convencional de se aplicar apenas um algoritmo para essa tarefa (que, provavelmente, pode ser o algoritmo mais dispendioso computacionalmente).

A partir dessa premissa, propõe-se, neste trabalho, uma solução baseada na aplicação combinada de diversos algoritmos criptográficos em partes específicas de uma mensagem SOAP, criando-se, assim, níveis criptográficos que serão definidos com base no grau de confidencialidade dos dados que compõem a mensagem SOAP em questão. Nesse caso, cada nível será criptografado com um determinado algoritmo criptográfico, tendo como base tanto a força de segurança desses algoritmos como o custo computacional dos mesmos (RODRIGUES; BRANCO, 2009). Essa técnica é definida neste trabalho como multicriptografia em *Web Services* ou multicriptografia SOAP.

Logo, a proposta da técnica de multicriptografia SOAP não reside apenas no fato de se garantir o fator segurança das aplicações *Web Services*, que necessitam implementar a segurança a nível de mensagem, mas, também, aprimorar os resultados de desempenho dessas aplicações ao implantar esse tipo de segurança, algo que não se observa nas soluções mais clássicas, como WS-Security (JENSEN et al., 2007), que almejam apenas garantir a segurança fim a fim das mensagens SOAP, não se preocupando com a questão do desempenho das aplicações-alvo.

Outro fator motivacional para a realização deste trabalho reside na possibilidade de se realizar a solução proposta como um *middleware* de segurança para *Web Services*, visando promover o benefício do desacoplamento do requisito segurança dessas aplicações. Isso tornaria o processo de desenvolvimento dos *Web Services* mais ágil, uma vez que os desenvolvedores não necessitariam lidar com determinadas questões ligadas à implementação

da segurança em nível de mensagem, tais como: escolha de algoritmos simétricos e assimétricos, compartilhamento de chaves, políticas de segurança, dentre outras questões pertinentes, pois todos esses aspectos seriam contemplados pelo *middleware*.

1.4 OBJETIVOS

1.4.1 Objetivo Geral

O objetivo geral deste trabalho consiste em propor uma solução de segurança, em nível de mensagem, para *Web Services* que satisfaça os requisitos de integridade e confidencialidade, promovendo também otimização de desempenho desses sistemas, ao se garantir esse tipo de segurança. Para tanto, objetiva-se aplicar, combinadamente, diversos algoritmos criptográficos para se criptografar os dados das mensagens SOAP, compartilhadas pelos *Web Services*. Tal solução será apresentada e discutida, detalhadamente, no Capítulo 4.

1.4.2 Objetivos Específicos

Para se atingir o objetivo geral, objetivos específicos foram delineados a fim de se definir uma abordagem que trate das duas principais causas que acarretam a deterioração do desempenho dos *Web Services*, ao realizarem o processo de criptografia em nível de mensagem, por serem, computacionalmente, as mais desgastantes para essa tarefa. A seguir, destacam-se as referidas causas atreladas aos objetivos específicos para se abordar cada uma delas:

a) com vista a tratar do problema da degradação do desempenho causado pela criptografia, no que diz respeito às partes de uma mensagem que devem ser efetivamente criptografadas (SOSNOSKI, 2009), propõe-se, aqui, a criptografia baseada na confidencialidade dos dados, de acordo com três níveis confidenciais: baixa, moderada e alta. O objetivo específico, nesse caso, consiste em estruturar as mensagens XML conforme a confidencialidade dos seus dados por meio de um XML *Schema*, denominado de *Levels of Confidentiality* (Níveis de Confidencialidade);

b) a segunda causa que provoca a referida degradação de desempenho está relacionada aos algoritmos criptográficos utilizados, assim como com o tamanho das chaves empregadas (RODRIGUES; BRANCO, 2009). Nesse aspecto específico, com base na análise do desempenho desses algoritmos associado ao tamanho das chaves, objetiva-se, neste trabalho, o uso combinado desses algoritmos, a fim de se garantir um equilíbrio entre os fatores de segurança e desempenho. Para tanto, tais algoritmos são atribuídos a cada nível de confidencialidade definido no XML *Schema Levels of Confidentiality,* considerando sua eficácia em termos, principalmente, de segurança e também de desempenho.

Outros objetivos específicos podem ser assim elencados:

a) conceituar e implementar a solução proposta como um *middleware* de segurança para *Web Services*, visando alcançar os benefícios alegados na seção 1.3;
b) realizar um estudo de caso que permita comprovar a validade da hipótese sustentada neste trabalho, ou seja, atestar que a aplicação de diversos algoritmos criptográficos para criptografar uma mensagem SOAP é mais eficaz que empregar apenas um único algoritmo para realizar a mesma tarefa.

1.5 METODOLOGIA DE PESQUISA

A metodologia de pesquisa adotada para o desenvolvimento deste trabalho está pautada em três pilares metodológicos:

a) **Revisão bibliográfica** – inicialmente, foi realizada uma revisão bibliográfica sobre o tema considerado, objetivando-se identificar evidências científicas que norteassem a formulação de uma proposta para resolução da problemática considerada. Essa revisão será devidamente apresentada no Capítulo 3;
b) **Pesquisa de Desenvolvimento –** esse tipo de pesquisa se caracteriza por utilizar, sistematicamente, conhecimentos adquiridos com o intuito de desenvolver um novo instrumento ou aperfeiçoar um já existente ou, até mesmo, aprimorar uma nova intervenção. Nesse caso, com base nas evidências científicas identificadas na revisão bibliográfica, definiu-se como proposta de trabalho o desenvolvimento de uma solução de segurança para *Web Services*, fundamentada na aplicação combinada de algoritmos criptográficos aos dados das mensagens SOAP. Essa etapa da pesquisa é descrita no Capítulo 4;

c) **Pesquisa Experimental (Experimentação)** – a experimentação consiste no conjunto de processos utilizados para se verificar hipóteses. Por conseguinte, no contexto deste trabalho, estipulou-se uma pesquisa experimental a fim de se avaliar a efetividade da hipótese de que a utilização de diversos algoritmos criptográficos é, de fato, mais eficaz do que utilizar apenas um determinado algoritmo para realizar o procedimento de criptografia em uma mensagem SOAP. Tal estudo experimental será apresentado no penúltimo Capítulo desta Dissertação, mais especificamente no 5º.

1.6 ORGANIZAÇÃO DO TRABALHO

Os demais capítulos que compõem este trabalho estão estruturados e organizados da seguinte forma:

a) **Capítulo 2 – Fundamentação Teórica**: explana a fundamentação teórica pertencente ao domínio deste trabalho;
b) **Capítulo 3 – Revisão Bibliográfica**: apresenta a revisão bibliográfica, analisando alguns trabalhos, direta e indiretamente, relacionados ao tema desta Dissertação, que deram subsídios para formulação da solução aqui proposta;
c) **Capítulo 4 – Arquitetura e Implementação do *Middleware* PerSec**: apresenta a arquitetura conceitual do *middleware* sugerido, bem como alguns detalhes técnicos sobre a sua prototipação;
d) **Capítulo 5 – Avaliação do *Middleware* PerSec**: expressa a realização de um estudo experimental que visa (além de analisar, de uma forma geral, o comportamento do *middleware* PerSec) comprovar a hipótese considerada neste estudo, que consiste na utilização de diversos algoritmos criptográficos, ao invés de apenas um algoritmo (prática usual) para realizar a criptografia das mensagens SOAP e, dessa forma, aprimorar o desempenho dos *Web Services* que implementam a segurança fim a fim, ou seja, a segurança em nível de mensagem;
e) **Capítulo 6 – Conclusão:** delineia as considerações finais, as limitações encontradas durante o desenvolvimento do trabalho, a produção científica auferida e ainda as perspectivas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este Capítulo tem como objetivo explanar os principais conceitos teóricos abordados no presente trabalho. Assim:

a) a primeira seção aborda a tecnologia dos *Web Services*, no que diz respeito aos seus principais componentes, e também o mecanismo de funcionamento dessas aplicações;
b) já a segunda seção apresenta uma visão geral sobre dois princípios básicos da segurança da informação – criptografia e assinatura digital – bem como os principais padrões XML que implementam esses princípios de segurança para as aplicações *Web Services*, ou seja, a XML *Encryption* e a XML *Signature*, respectivamente;
c) a última seção encerra o Capítulo elencando algumas considerações finais sobre todo o conteúdo explanado.

2.1 VISÃO GERAL SOBRE *WEB SERVICES*

De maneira geral, a tecnologia *Web Services* é o meio mais difundido para uma implementação SOA. Esses serviços podem ser compreendidos como determinadas funcionalidades de aplicações definidas em XML (*Extensible Markup Language*) que são disponibilizadas para outras aplicações, objetos ou, até mesmo, bases de dados na forma de serviços. Considerando esse panorama, uma determinada aplicação realiza uma solicitação formatada em XML à outra aplicação qualquer, por meio de uma rede (a *Internet*, na maioria dos casos), recebendo como resposta os dados solicitados, também no formato XML – ressalte-se que os padrões estabelecidos para *Web Services* determinam tanto a forma das mensagens (em termos estruturais) como a maneira que as mesmas serão enviadas. Além disso, tais padrões especificam questões, como o mapeamento de conteúdo das mensagens de forma interna e externa em relação aos programas que implementam o serviço, e estipulam meios para publicar e descobrir interfaces de *Web Services*.

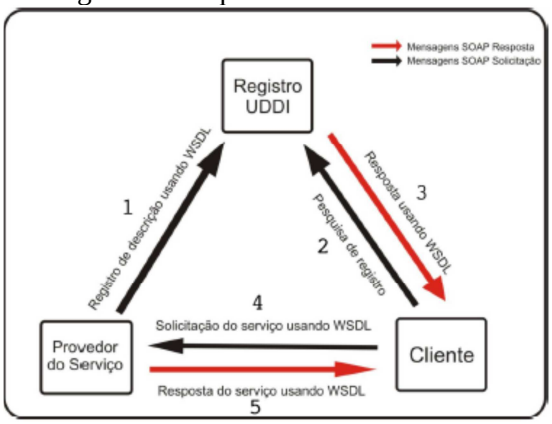
No que diz respeito a sua execução, os *Web Services* podem ser aplicados em plataformas, como computadores *desktops* e até mesmo *handhelds*. Além disso, é possível usá-los para integração B2B (*Business-to-Business)*, o que significa conectar aplicações executadas em diversas organizações em um mesmo aplicativo.

Seguindo a linha evolutiva dos *Web Services*, muitas APIs (*Application Programming Interface*), em diversas linguagens de programação, foram criadas para a tarefa de

desenvolvimento dessas aplicações, culminando em padrões de desenvolvimento, como o QoS (*Quality of Service*) e controle de segurança.

A Figura 2 demonstra, basicamente, a estrutura de um *Web Service*. Tal estrutura é formada por um cliente, um *Web Service*, além do diretório de registro de serviços. Esquematicamente, o cliente seleciona um serviço desejado de acordo com a disponibilidade deste no diretório de registro de serviços. Dessa forma, o cliente tem acesso a um arquivo contendo a descrição, em uma linguagem apropriada, da interface do serviço selecionado. A partir dessa descrição, o cliente tem a possibilidade de invocar o serviço em questão. Caso isso seja executado, após realizar o processamento da solicitação, o serviço, como resposta, envia uma mensagem contendo o resultado da solicitação ao cliente.

Figura 2 – Arquitetura básica dos Web Services



Fonte: Adaptado de Souza (2010).

A arquitetura dos *Web Services*, ilustrada na Figura 2, está baseada nos seguintes padrões:

a) XML: padrão utilizado para formatar as mensagens transitadas entre o cliente e o *Web Services*;
b) SOAP: padrão responsável por encapsular, em forma de pacotes, as mensagens XML;
c) HTTP: padrão, geralmente, empregado para executar o transporte dos pacotes SOAP;
d) WSDL: padrão adotado para descrever a interface do serviço;
e) UDDI: padrão estabelecido para armazenar a descrição do serviço.

Nas próximas subseções, os padrões mais diretamente abordados neste trabalho serão discutidos de forma detalhada.

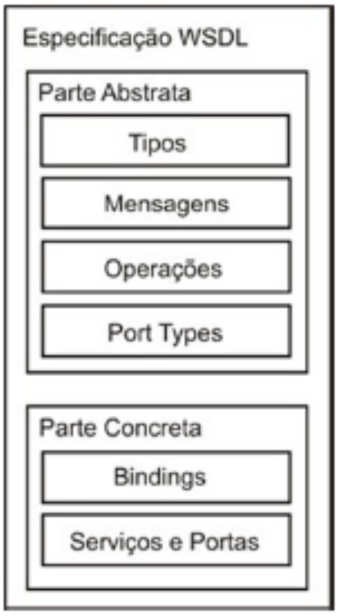
2.1.1 Web Service Definition Language – WSDL

A especificação WSDL foi proposta e desenvolvida inicialmente pelo consórcio entre as empresas IBM, Microsoft e Ariba, por meio da unificação de três propostas: *Microsoft's SOAP Contract Language, Service Description Language e o Network Accessible Service Specification Language*. Com o passar do tempo, a WSDL foi evoluindo, culminando, atualmente, na sua versão 2.0 (CANYANG; BOOTH, 2007).

A finalidade da WSDL é definir um formato universal para descrever e publicar informações sobre os serviços ofertados pelos *Web Services* na *Web*. Para tanto, a WSDL faz uso de elementos baseados em um ou mais esquemas XML, cuja finalidade é descrever os dados envolvidos em uma comunicação com o propósito de torná-los compreensíveis para todas as partes nela envolvidas.

De uma forma geral, a especificação WSDL pode ser decomposta em duas partes: uma, abstrata, e outra, concreta. A primeira parte considerada se encarrega de definições do *port type* – uma coleção lógica de operações relacionadas, sendo que cada operação, por sua vez, define uma troca de mensagens simples (conceito empregado para representar uma unidade de comunicação com um *Web Service*, representando os dados trocados em uma transmissão lógica simples). Já a parte concreta do WSDL tem como objetivo estipular protocolos de conexão, além de outras informações, como ilustra a Figura 3 (ALONSO, 2003).

Figura 3 – Especificação do serviço WSDL



Fonte: Adaptado de Souza (2010).

É possível afirmar que a WSDL, por se tratar de uma linguagem genérica para descrição de serviços, pode ser aplicada independentemente do sistema, o que possibilita uma diversidade de configurações.

2.1.2 Simple Object Access Protocol – SOAP

O protocolo SOAP pode ser considerado como uma das tecnologias mais relevantes das que integram o padrão *Web Services*, visto que estes não podem subsistir sem uma forma abstrata de representar seus dados e publicar suas definições de interface. Entretanto, sua função essencial é, simplesmente, obter os dados de um local para outro em uma rede, permitindo, assim, que o transmissor e o receptor de documentos XML realizem uma comunicação efetiva. Para tanto, o SOAP, analogamente, atua como uma pequena extensão do HTTP para que este suporte as mensagens XML (ALONSO, 2003).

De uma forma geral, o padrão SOAP estabelece uma maneira estruturalmente escrita de organizar as mensagens XML para que estas possam ser, compreensivamente, trocadas na rede, como aponta Alonso (2003) ao definir esta tecnologia como 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.

Para finalizar esta subseção, listam-se, a seguir, algumas propriedades essenciais do protocolo de comunicação SOAP (GUDGIN, 2007; SNELL, 2001):

a) é desprovido de estado, ou seja, não possui informações sobre o estado do *Web Service*;
b) ignora a semântica das mensagens que transporta;
c) suporta aplicações fracamente acopladas, que são caracterizadas por interagir por meio da troca de mensagens assíncronas e em mão única;
d) toda complexidade que extrapole o padrão SOAP, como mensagens síncronas em mão dupla ou, até mesmo, interação estilo RPC (*Remote Procedure Call*) (MICROSOFT, 2015), necessitará da associação do SOAP com outros protocolos ou *middleware*s que implementem as propriedades requeridas;
e) a troca de informações em SOAP é realizada via mensagens, que são encapsuladas pelo protocolo em questão como se fosse um envelope, sendo que, estruturalmente, esse envelope possui duas partes: um “cabeçalho” (opcional, podendo ou não estar

presente na mensagem) e um “corpo” (obrigatório). Tanto o cabeçalho como o corpo são compostos por múltiplas subpartes designadas “blocos do cabeçalho” ou “blocos do corpo”. Em todo caso, um bloco de cabeçalho, ou um bloco de corpo, pode ser compreendido como qualquer “filho” de primeiro nível do elemento cabeçalho (ou corpo) de uma mensagem;

f) para tornar o SOAP funcional, é preciso, primeiramente, especificar como o mesmo será transportado pela rede, pois nenhum protocolo é imposto, apesar de, em geral, associarem-no ao HTTP. De qualquer forma, se faz necessário definir a forma como a mensagem SOAP será empacotada e também quais regras do protocolo de transporte serão aplicadas (no caso da associação com HTTP, os modos mais comumente utilizados são o HTTP GET e o HTTP POST);

g) a identificação do destinatário da mensagem em termos de endereço é um item indispensável. Desta forma, uma parte da mensagem SOAP é interpretada e posteriormente inserida no protocolo de transporte de rede para que o mesmo entregue a mensagem ao destinatário correto;

h) no roteamento de mensagens SOAP, um *path* de mensagens contém a descrição dos nós pelos quais a mensagem deve trafegar. A interpretação desse *path* é que possibilita que o roteamento seja executado.

As próximas subseções abordarão os princípios fundamentais sobre a segurança computacional, que visam garantir aspectos de confidencialidade, integridade e autenticidade às informações, bem como as especificações XML empregadas para se implementar esses fundamentos da segurança da informação às mensagens SOAP.

2.2 VISÃO GERAL SOBRE SEGURANÇA COMPUTACIONAL

Há três princípios básicos para se garantir a segurança da informação (ALBUQUERQUE, 2002; KRAUSE, 1999):

a) Confidencialidade – a informação somente pode ser acessada por pessoas explicitamente autorizadas (o que seria a proteção de sistemas de informação para impedir que pessoas não autorizadas tenham acesso);

b) Disponibilidade – a informação deve estar disponível no momento em que a mesma for necessária;

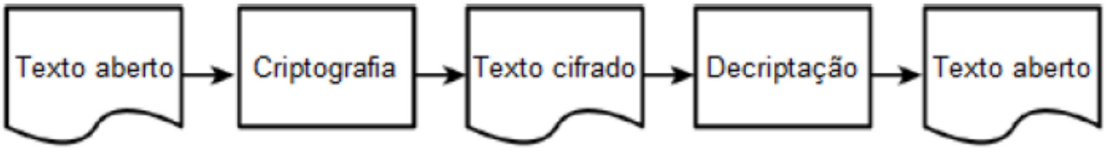
c) Integridade – a informação deve ser recuperada em sua forma original (no momento em que foi armazenada), o que seria a proteção dos dados ou informações contra modificações intencionais ou acidentais não autorizadas.

O item “Integridade” não pode ser confundido com confiabilidade do conteúdo (significado) da informação. Uma informação pode ser imprecisa, mas deve permanecer íntegra (não sofrer alterações por pessoas não autorizadas). Algumas técnicas mais comumente utilizadas para se atender aos princípios básicos da segurança da informação descritos anteriormente são a criptografia e a assinatura digital.

2.2.1 Criptografia

A criptografia surgiu com o advento da escrita, da necessidade que o homem sentiu em esconder informações consideradas sigilosas. Esta técnica hoje em dia é a base tecnológica para problemas de segurança em comunicações e em computação. De acordo com Burnett (2002), a Criptografia converte dados legíveis em algo sem sentido, ilegível. Sendo que os dados originais poderão ser recuperados posteriormente a partir desses dados sem sentido. De forma mais detalhada, a criptografia consiste em uma série de métodos e técnicas empregadas para criptografar dados compreensíveis, aplicando-se um algoritmo criptográfico por meio de uma chave como parâmetro, convertendo os dados originais (também conhecidos como texto aberto ou, ainda, texto claro ou texto simples) em dados incompreensíveis (conhecidos também como texto cifrado, cifra ou texto-código). Por outro lado, o receptor tem condições de decifrar esses dados cifrados, o que pode ser definido como o processo inverso da criptografia (decriptação), obtendo, dessa maneira, os dados no seu formato original (TANENBAUM, 2003; MORENO et al., 2005). A Figura 4 ilustra apropriadamente o mecanismo de funcionamento desse processo.

Figura 4 – Esquema geral da criptografia



Fonte: Adaptado de Rosenberg e Remy (2004).

Os algoritmos envolvidos no processo de criptografia podem ser interpretados como uma sequência de instruções matemáticas criadas especificamente para criptografar e decifrar informações consideradas sigilosas. Entretanto, esses algoritmos, para realizarem suas

funções, necessitam ser parametrizados com a chave correta – que, no contexto computacional, significa um conjunto de *bits* que determinam o tamanho da mesma.

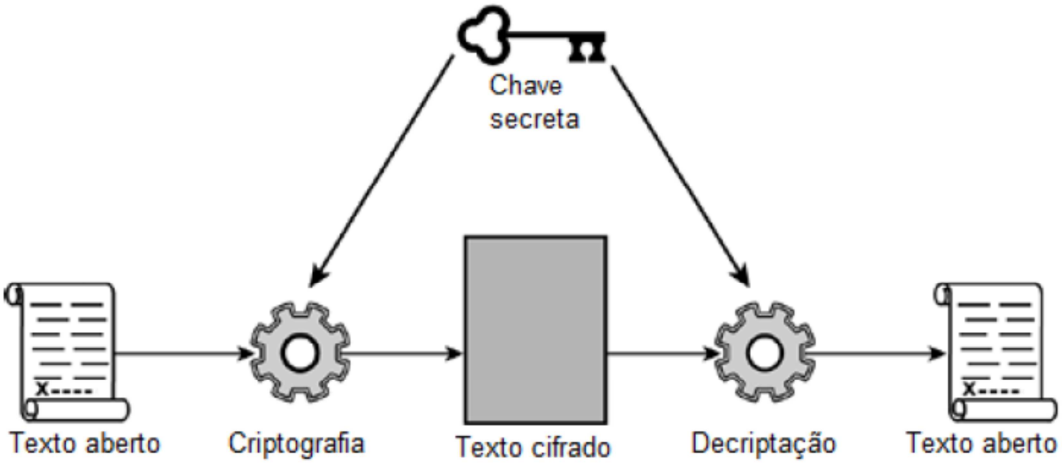
É importante ressaltar que, baseados no princípio de Kerckhoff, os algoritmos criptográficos devem ser mantidos em domínio público, pois, segundo esse princípio, o esforço exigido para desenvolver, testar e instalar um novo algoritmo, em caso de total comprometimento de um algoritmo anterior, torna impraticável mantê-lo em sigilo. Segundo esse mesmo princípio, apenas as chaves devem ser secretas, portanto, os algoritmos são sempre divulgados publicamente à comunidade. Desta forma, o sigilo das informações é garantido efetivamente pela chave, principalmente quando esta possui um tamanho considerável, uma vez que quanto maior o seu comprimento, mais forte será a criptografia para um mesmo algoritmo (TANENBAUM, 2003).

Outro aspecto importante a ser considerado no que diz respeito à chave criptográfica é que, conforme o seu tipo, o processo de criptografia pode ser categorizado como: criptografia de chave simétrica ou criptografia de chave pública.

2.2.1.1 Criptografia de Chave Simétrica

A criptografia de chave simétrica (ou criptografia de chave privada) é caracterizada por realizar seus procedimentos de criptografia e decriptação por meio de apenas uma chave, ou seja, a mesma chave é compartilhada tanto pelo emissor como pelo receptor. Assim, manter a chave em absoluto sigilo é um pré-requisito fundamental para se garantir a segurança das informações envolvidas na criptografia. A Figura 5, a seguir, ilustra esse processo, no qual o emissor criptografa o texto aberto em texto cifrado, fornecendo a chave secreta compartilhada como parâmetro para o algoritmo criptográfico em questão. Por outro lado, após receber a mensagem criptografada transmitida, o receptor pode então decriptá-la, utilizando a mesma chave secreta como parâmetro para, então, obter as informações no seu formato original.

Figura 5 – Criptografia de chave simétrica



Fonte: Adaptado de Rosenberg e Remy (2004).

A criptografia de chave simétrica apresenta como característica importante a rapidez dos seus algoritmos, além do fato de poderem ser aplicados a mensagens de tamanhos arbitrários (ROSENBERG; REMY, 2004). Porém, alguns aspectos desvantajosos podem ser observados nesse tipo de criptografia, tais como: (i) – a dificuldade em gerenciar uma chave compartilhada (devido à necessidade de enviá-la para todos os usuários autorizados) e (ii) – a necessidade de mantê-la em segredo para usuários não autorizados, a fim de garantir a segurança das informações (MORENO et al., 2005).

No que diz respeito aos algoritmos criptográficos de chave simétrica, os mais utilizados são:

a) DES (*Data Encryption Standard*) – este algoritmo foi criado pela IBM em 1977 e, logo em seguida, definido oficialmente como padrão para informações não confidenciais pelo governo norte-americano, sendo também adotado amplamente pelo setor comercial. Uma característica essencial do DES diz respeito ao tamanho da sua chave, mais precisamente 64 *bits*. Por meio dela, o DES criptografa texto aberto em blocos de 64 *bits*, gerando 64 *bits* de texto cifrado (TANENBAUM, 2003). Entretanto, apesar do seu sucesso inicial, as chaves do DES se tornaram frágeis diante do enorme poder de processamento dos computadores atuais, obsoletando esse algoritmo por questões de insegurança e tornando-o não recomendado para a criptografia de dados muito sigilosos (KANNEGANTI; CHODAVARAPU, 2008);
b) 3DES (*Triple Data Encryption Standard*) – o 3DES surgiu como proposta (do governo norte-americano) de atualização do DES, objetivando sanar a deficiência deste último. Para tanto, o 3DES executa três vezes seguidas o algoritmo DES de 64 *bits*, o que proporciona um aumento significativo no tamanho da chave, ou seja, de 64 *bits* para

192 *bits*. Esquematicamente, o mecanismo de funcionamento do 3DES realiza a criptografia dos dados considerando as seguintes etapas: (i) – criptografa os dados por meio do algoritmo DES utilizando uma chave de 64 *bits*; (ii) – executa o algoritmo DES no modo de decriptação, utilizando uma segunda chave, sobre a saída da primeira etapa já descrita; (iii) – a terceira e última etapa consiste em uma nova execução do algoritmo DES no modo de criptografia, utilizando uma terceira chave, sobre a saída da segunda etapa de criptografia (KUROSE; ROSS, 2009). A lentidão desse processo pode ser citada como uma desvantagem do 3DES em relação a outros algoritmos de criptografia simétrica, como o AES, descrito a seguir;

c) AES (*Advanced Encryption Standard*) – este algoritmo teve origem em um concurso de criptografia patrocinado pelo NIST (*National Institute of Standards and Technology*), realizado em 2001, cujo objetivo era eleger um algoritmo gabaritado a ser o sucessor do 3DES. Elegeu-se, na ocasião, o algoritmo Rijndael (DAEMEN; RIJMEN, 2000), que passou a ser chamado de AES desde então. Tal algoritmo se caracteriza por apresentar chaves cuja extensão pode variar entre 128, 192 ou 256 *bits* e bloco de 128 *bits* (FELDHOFER et al., 2004). Para comprovar sua eficiência, segundo análise do NIST, uma máquina capaz de quebrar o DES de 64 *bits* em 1 segundo necessitaria, estimativamente, de 149 milhões de anos para quebrar o AES de 128 *bits* (KUROSE; ROSS, 2009).

A próxima subseção apresenta as principais características de outra forma de criptografia: a criptografia de chave pública.

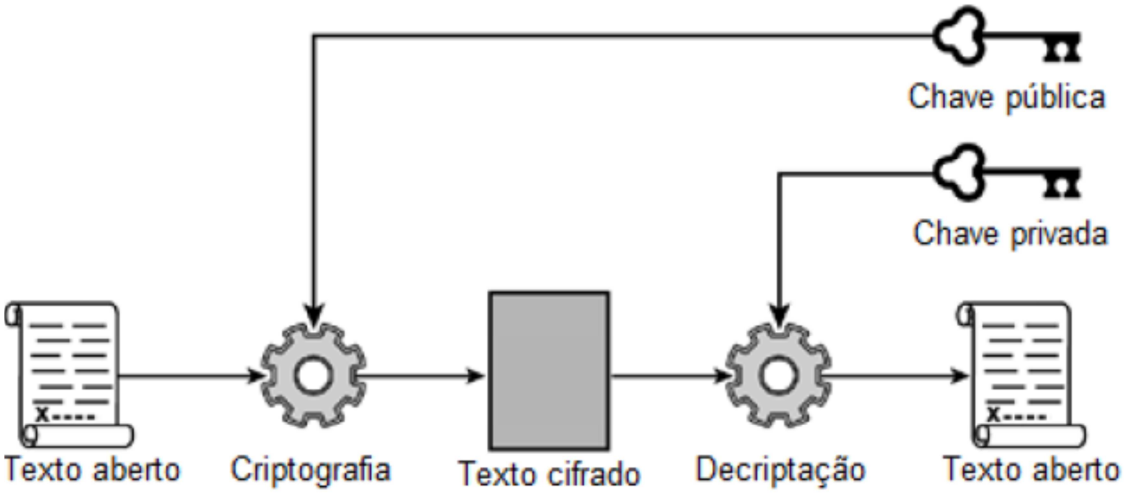
2.2.1.2 Criptografia de Chave Pública

Um grande impasse da criptografia de chave simétrica diz respeito à distribuição de suas chaves, uma vez que, para se estabelecer uma comunicação entre as partes envolvidas, as mesmas devem estar de acordo com a chave compartilhada, mas, para tanto, é preciso se certificar de que a comunicação estabelecida é supostamente segura. Diante disso, em 1976, os pesquisadores Whitfield Diffie e Martin Hellman, ambos da Universidade de Stanford, desenvolveram o algoritmo *Diffie-Hellman Key Exchange* (DIFFIE; HELLMAN, 1976) como uma proposta para resolver o referido impasse. Tal algoritmo apresenta, como característica fundamental, chaves de criptografia e decriptação distintas. Além disso, a chave de decriptação não pode ser derivada da chave criptográfica (TANENBAUM, 2003).

Deste modo, a criptografia de chave pública, ou criptografia de chave assimétrica, como também é conhecida, se caracteriza por empregar dois tipos de chaves: chave privada e chave pública, sendo que ambas podem ser utilizadas para criptografar os dados. Em contrapartida, a decriptação só é possível com a chave oposta, por exemplo: caso a criptografia seja realizada com a chave pública, a decriptação deverá ser executada com a respectiva chave privada. No entanto, para a efetiva aplicação desse algoritmo, é de fundamental importância manter a chave privada em sigilo, compartilhando apenas a chave pública para aqueles que desejarem se comunicar (ROSENBERG; REMY, 2004).

A Figura 6 ilustra o mecanismo de funcionamento da criptografia de chave pública. No contexto desta ilustração, o emissor da mensagem utiliza a chave pública do receptor para criptografar o texto aberto em texto cifrado. Em seguida, assim que o receptor recebe o texto cifrado resultante, o mesmo, por meio da sua chave privada, realiza o processo de decriptação do texto cifrado no texto original.

Figura 6 – Criptografia de chave pública



Fonte: Adaptado de Rosenberg e Remy (2004).

Dentre os diversos algoritmos que podem ser empregados na criptografia de chave pública, destaca-se o algoritmo RSA (*Rivest, Shamir, Adleman*), de forma predominante (KUROSE; ROSS, 2009). Ele é reconhecido como um algoritmo forte, entretanto, o fato da exigência de chaves de, no mínimo, 1024 *bits* (para que haja um nível de segurança equivalente ao nível de segurança apresentado pelos algoritmos de chave simétrica de 128 *bits*) consiste em uma desvantagem para o mesmo, pois o torna bastante lento (TANENBAUM, 2003).

Por conta dessa lentidão, a tarefa de criptografar uma grande quantidade de dados não é indicada para o RSA. Por outro lado, o mesmo se mostra muito eficaz quando empregado para a distribuição de chaves. Assim, praticamente, boa parte dos sistemas emprega um algoritmo de chave pública, como o RSA, apenas para criptografar e distribuir chaves secretas compartilhadas, as quais serão parametrizadas a algum algoritmo de chave simétrica, tais como o 3DES ou o AES, para que assim seja possível executar a criptografia dos dados em questão (KOCHER et al., 2004).

2.2.2 Assinatura Digital

A assinatura digital serve, basicamente, para autenticar documentos legais, financeiros, dentre outros. Em decorrência disso, seus princípios se equivalem aos princípios das assinaturas por escrito, ou seja, devem ser verificáveis, não falsificáveis e incontestáveis. De forma mais minuciosa, uma assinatura é **verificável** quando é possível provar que ela, realmente, pertence à determinada pessoa; é **não falsificável** quando é possível afirmar que somente determinada pessoa poderia ter assinado um documento qualquer e, consequentemente, é **incontestável** pelo fato de o proprietário da assinatura não poder repudiá-la (KUROSE; ROSS, 2009).

Uma forma de contemplar todos esses princípios é por meio da criptografia de chave pública, aplicando o algoritmo RSA, por exemplo. A criptografia pode ser realizada com a chave privada e a decriptação, com a chave pública, sendo que a recíproca também é verdadeira, ou seja, a criptografia pode ser realizada por meio da chave pública e decriptação, por meio da chave privada. Desta forma, é possível verificar a autenticidade da assinatura, seguindo a lógica de que se o emissor criptografa uma determinada mensagem com sua chave privada, enviando-a em seguida para o receptor, este, por sua vez, deverá usar a chave pública do emissor para confirmar se foi, realmente, enviada pelo emissor (SOSNOSKI, 2009). Mas, para que este procedimento seja efetivamente seguro, é indispensável que se mantenha a chave privada em sigilo.

Em termos práticos, por conta da lentidão dos algoritmos de chave pública, a técnica de assinatura digital assina apenas um sumário de mensagem (*message digest*), ou resumo de mensagem, que é gerado aplicando-se uma função de *hash* na mensagem (MORENO et al., 2005). Em linhas gerais, esse tipo de função captura um fragmento textual da mensagem, computando uma cadeia de *bits* com base nesse fragmento, que será a saída gerada. Tal

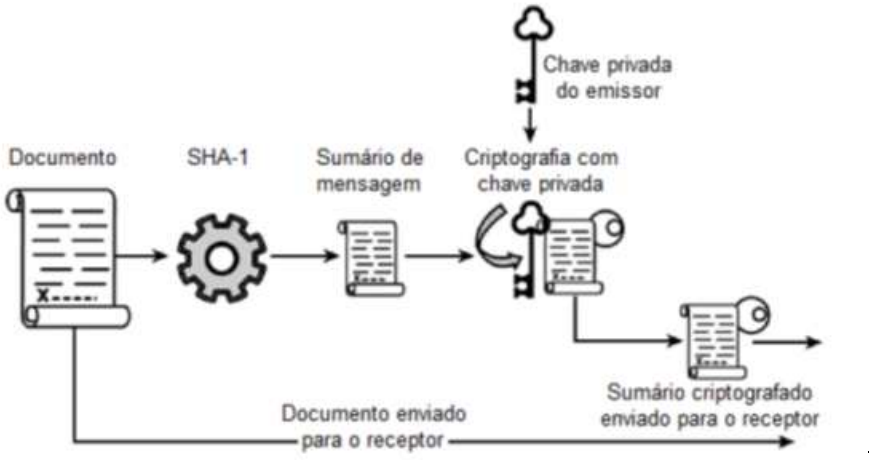
processo combina criptografia de chave pública com a função de *hash*, como demonstra a Figura 7 adiante. Decompondo-se o processo de assinatura digital em etapas:

1. aplica-se a função de hash, como o SHA-1 (*Secure Hash Algorithm 1*), no documento contendo texto aberto, resultando em um sumário de mensagem de comprimento fixo;
2. o sumário de mensagem é criptografado com a chave privada do emissor;
3. na sequência, a mensagem, juntamente com o seu sumário, devidamente, criptografado com a chave privada do emissor, é encaminhada para o receptor.

O receptor, por sua vez, ao receber o documento, realiza a verificação da assinatura digital, tanto para confirmar a identidade do emissor como para checar se o documento não sofreu alterações, como ilustra a Figura 7. Tal processo é composto pelas seguintes etapas:

1. primeiramente, é entregue ao receptor o documento contendo texto aberto, bem como o sumário de mensagem criptografado;
2. o receptor obtém a chave pública do emissor, não importando se no mesmo instante em que recebeu a mensagem, ou após;
3. executa-se novamente a mesma função de *hash* (o SHA-1), utilizada pelo emissor, tendo como parâmetro o mesmo documento original em texto aberto. Como este algoritmo é homogêneo, considerando todas as plataformas, o receptor, ao executá-lo, obtendo o mesmo resultado, pode considerar o documento inalterado.

Figura 7 – Geração de Assinatura Digital



Fonte: Adaptado de Rosemberg e Remy (2004).

4. o receptor, então, faz uso da chave pública do emissor para decriptar o sumário de mensagem. Caso a operação seja bem-sucedida e considerando a chave pública do emissor confiável, o receptor atestará que o documento em questão foi, realmente, enviado pelo referido emissor;

5. por fim, ocorre uma comparação *bit* a *bit* entre o sumário de mensagem computado localmente a partir do documento original e o sumário de mensagem recebido e já decriptado. Se eles forem exatamente iguais, a assinatura é considerada válida.

As próximas subseções apresentam as duas especificações de segurança XML que implementam os conceitos de segurança tratados nesta seção, ou seja, a especificação XML *Encryption* para implementar o processo de criptografia e a especificação XML *Signature* para implementar o processo de assinatura digital.

2.2.3 XML Encryption

A especificação XML *Encryption* foi criada pela W3C (W3C, 2002), tendo como objetivo criptografar dados e representá-los de forma estruturada em um documento XML, garantindo, desta maneira, o princípio de confidencialidade aos mesmos. Essa especificação procura prover uma segurança fim a fim para aplicações que necessitam trocar dados em formato XML, sem correr o risco de que os mesmos tenham seu conteúdo revelado e utilizado indevidamente por *hackers* mal-intencionados (SIDDIQUI, 2002). Esse tipo de segurança é de vital importância para aplicações que requerem uma comunicação segura de dados formatados em XML, pois procura garantir que tais dados não sejam indevidamente revelados e usufruídos por terceiros (SIDDIQUI, 2002).

Uma característica marcante da especificação XML *Encryption* diz respeito a sua capacidade de criptografar os dados em diferentes níveis de granularidades, ou seja, criptografar um dado elementar ou um conjunto deles, em um documento XML qualquer, permitindo que o restante dos dados do documento em questão permaneçam íntegros. Tal forma criteriosa de se aplicar a criptografia representa uma grande vantagem dessa especificação, levando-se em conta o fato do alto custo computacional da criptografia XML, principalmente em termos de tempo de CPU, isto é, tempo de processamento (NAGAPPAN, 2003). No caso em tela, a solução proposta neste estudo estabelece o critério de se criptografar os dados que compõem um documento XML, de acordo com o nível de confidencialidade dos mesmos: aplica-se um algoritmo criptográfico para cada nível confidencial proposto, conforme a classificação desses algoritmos em termos de segurança e eficiência, como já discutido em outras seções, numa tentativa de aprimorar o processo de criptografia XML referente ao tempo de execução desse processo.

A Figura 8 ilustra uma estrutura empregada pelo XML *Encryption* para representar os dados resultantes da criptografia realizada sobre um determinado elemento.

Figura 8 - Estrutura do XML Encryption

```
01.  <xenc:EncryptedData
02.      xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
03.      Type="http://www.w3.org/2001/04/xmlenc#Element">
04.    <xenc:EncryptionMethod
05.        Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
06.    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
07.      <ds:KeyName>Key</ds:KeyName>
08.    </ds:KeyInfo>
09.    <xenc:CipherData>
10.      <xenc:CipherValue>ni320jas2...</xenc:CipherValue>
11.    </xenc:CipherData>
12.  </xenc:EncryptedData
```

Fonte: Adaptado de Rodrigues e Branco (2009).

De forma geral, pode-se explicar tal estrutura da seguinte forma: (i) – Na linha 1, observa-se o elemento raiz *EncriptedData*, responsável por armazenar informações, tais como o *namespace* na linha 2, além de especificar que apenas o elemento, destacado na linha 3, está sendo criptografado; (ii) – o elemento *EncriptedData* é pai de três elementos-filhos: *EncryptionMethod* (especifica o algoritmo empregado para a criptografia, o 3DES, no caso da ilustração proposta, detalhe que pode ser observado na linha 5); *KeyInfo* (responsável apenas por armazenar informações referentes à chave); e *CipherData* (o qual contém o resultado final do processo de criptografia sobre o elemento em questão, armazenando-o no elemento *CipherValue*, destacado na linha 10).

A decriptação, por sua vez, é realizada por meio do seguinte processo: (i) – obtém-se o texto cifrado a partir do elemento que o contém, ou seja, o elemento *CipherValue* (linha 10); (ii) – identifica-se o algoritmo utilizado contido no elemento *EncryptionMethod* (linha 5); (iii) – obtêm-se as informações referentes à chave contidas no elemento *KeyInfo* (linha 6) e, finalmente; (iv) – verifica-se o elemento que foi criptografado (linha 3). Desta forma, coletando-se todas estas informações, é possível realizar a decriptação dos dados.

2.2.4 XML *Signature*

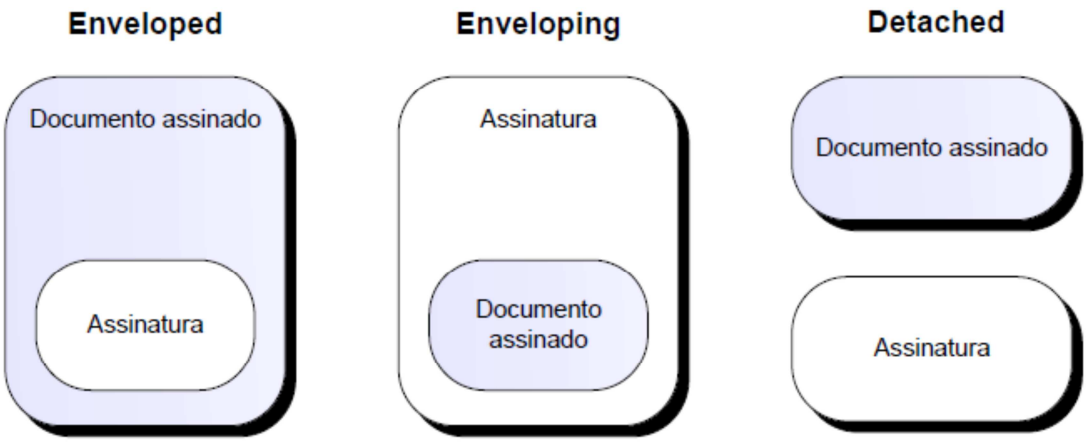
A especificação XML *Signature* é um padrão da W3C (W3C, 2008b), que tem como objetivo garantir a integridade e autenticidade de documentos XML, dentre outros, por meio de um processo que visa gerar e validar assinaturas digitais expressas em XML (MOGOLLON, 2008).

Assim como a XML *Encryption*, a especificação XML *Signature* também permite que sua ação se dê apenas em partes do documento considerado, ou seja, a assinatura de partes específicas do documento XML (YUE-SHENG, 2009). Essa característica da XML *Signature*

possibilita que um documento XML sofra alterações nas partes não assinadas sem, contudo, impactar a parte assinada no sentido de invalidá-la. Isto se constitui em uma grande vantagem, pois, se a especificação XML *Signature* previsse apenas a possibilidade de assinar o documento XML inteiro, qualquer alteração nos dados do documento se refletiria em uma invalidação da assinatura.

As assinaturas fornecidas pelo padrão XML *Signature* podem variar em três tipos, conforme ilustra a Figura 9. Os dois primeiros tipos ilustrados – o *Enveloped* (utilizado em *Web Services*) e o *Enveloping* – se caracterizam por acompanhar o documento XML que os mesmos assinam. O tipo *Detached*, por sua vez, funciona de forma contrária aos dois primeiros tipos, isto é, separa a assinatura dos dados assinados, mantendo apenas uma referência destes dados na estrutura XML da assinatura (NAGAPPAN, 2003).

Figura 9 – Tipos de assinaturas do XML Signature



Fonte: Adaptado de Nordbotten (2009).

A estruturação de uma assinatura baseada na especificação XML *Signature* é ilustrada na Figura 10 a seguir.

Figura 10 - Estrutura do XML Signature

```
01.  <ds:Signature>
02.    <ds:SignedInfo>
03.      <ds:CanonicalizationMethod
04.          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
05.      <ds:SignatureMethod
06.          Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
07.      <ds:Reference URI="...">
08.        <ds:Transforms .../>
09.        <ds:DigestMethod
10.            Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
11.        <ds:DigestValue>pkKoUrEWaYhQhJKpfx9...</ds:DigestValue>
12.      </ds:Reference>
13.    </ds:SignedInfo>
14.    <ds:SignatureValue>nnzElamjC1N aMA0...</ds:SignatureValue>
15.    <ds:KeyInfo>...</ds:KeyInfo>
16.  </ds:Signature>
```

Fonte: Adaptado de Rodrigues e Branco (2009).

Na primeira linha, encontra-se o elemento *Signature*, considerado o elemento-raiz dessa estrutura e pai dos elementos *SignedInfo*, *SignatureValue* e *KeyInfo*, que podem ser explicados, resumidamente, da seguinte forma:

a) *SignedInfo* – este elemento, localizado na linha 2, determina certos detalhes intrínsecos do processo de assinatura por meio de elementos-filhos, tais como: o elemento *CanonicalizationMethod* (disposto na linha 3) define qual foi o algoritmo empregado na normalização do documento (como pode ser observado na linha 4);
b) *SignatureMethod* – o elemento *SignatureMethod*, localizado na linha 5, por sua vez, informa qual algoritmo é aplicado na assinatura (que mais precisamente, conforme pode-se observar na linha 6, é o resultado da combinação do algoritmo de criptografia de chave pública RSA com a função de *hash* SHA-1);
c) *Reference* – o elemento *Reference*, que se encontra na linha 7, além de referenciar os dados assinados, contém outras informações, tais como: o algoritmo empregado para criar o sumário de mensagem, que, na ilustração proposta, mais precisamente na linha 10, percebe-se tratar do algoritmo SHA-1; além do valor de sumário propriamente dito, expresso na linha 11;
d) *SignatureValue* – este elemento, descrito na linha 14, apenas expressa o valor da assinatura;
e) *KeyInfo* – por fim, este último elemento, localizado na linha 15, completa a estrutura da assinatura em questão, fornecendo informações a respeito da chave empregada na verificação da assinatura.

Em contrapartida, cabe ao receptor executar o processo de verificação de integridade do documento XML, efetuando, localmente, o cálculo do sumário da mensagem, comparando-o em seguida com o sumário de mensagem presente na mesma. Em caso de compatibilidade dos sumários, considera-se como verdadeira a validade da assinatura, consequentemente, validando-se também a integridade do documento.

2.3 CONSIDERAÇÕES FINAIS

Neste Capítulo, buscou-se apresentar toda a fundamentação teórica envolvida no presente trabalho, explanando os:

a) conceitos fundamentais dos *Web Services*, em que se destacam os padrões mais relevantes para o desenvolvimento do estudo em questão;

b) conceitos básicos referentes à segurança geral em redes de computadores e as técnicas mais clássicas para a garantia da segurança, tais como a criptografia e a assinatura digital;
c) padrões de segurança XML que, geralmente, são empregados para se implementar os mecanismos de criptografia e assinatura digital para as aplicações *Web Services*, mais precisamente: a XML *Encriptyon* e a XML *Signature*, discutindo-se sobre suas principais características e mecanismo de funcionamento.

O próximo Capítulo apresenta a revisão bibliográfica, cujo objetivo é, por meio da análise dos diversos trabalhos correlatos, identificar subsídios que sirvam como diretrizes para a formulação de uma nova proposta de solução para a problemática considerada neste estudo.

3 REVISÃO BIBLIOGRÁFICA

A revisão bibliográfica empregada nesta Dissertação seguiu diretrizes estabelecidas em um protocolo de revisão acerca da literatura acadêmica sobre trabalhos direta e indiretamente relacionados ao tema deste trabalho. A revisão é descrita em maiores detalhes nas subseções que compõem este capítulo, da seguinte forma:

a) na seção 3.1, o protocolo utilizado é descrito, apresentando-se as questões de pesquisas investigadas; a definição dos termos extraídos das referidas perguntas (que serviram como parâmetros na execução da revisão); as regras elaboradas para a seleção e avaliação da qualidade dos estudos obtidos como resultado desta revisão; as fontes de pesquisas consideradas, culminando na catalogação de todos os trabalhos validados durante a execução do protocolo proposto;
b) na seção 3.2, todos os trabalhos catalogados são apresentados de forma analiticamente resumida, a fim de se detectar evidências científicas úteis para a definição de uma nova solução para a problemática abordada neste estudo;
c) a última seção, por sua vez, encerra o Capítulo delineando algumas considerações finais acerca do resultado obtido com a execução do protocolo de pesquisa proposto, ou seja, da análise dos trabalhos relacionados oriundos desse protocolo de revisão.

3.1 PROTOCOLO DE REVISÃO

3.1.1 Questões de Pesquisa

A elaboração de uma questão de pesquisa exige um processo criativo por parte dos seus desenvolvedores, tendo como interesse um objeto de estudo em particular. Vale ressaltar que as ideias norteadoras das questões de pesquisa, inicialmente, são vagas e, portanto, requerem uma análise acurada para que evoluam, de modo gradual, para projetos de pesquisa bem estruturados e consolidados. Dessa forma, uma investigação científica requer a formulação de uma ou mais questões de pesquisa, de forma clara e objetiva (SAMPIERI et al., 2006).

Logo, esta revisão da literatura iniciou-se por meio da seguinte questão: 1ª – Como garantir a segurança fim a fim em aplicações *Web Services*?

Na busca pela sua resposta, os primeiros trabalhos correlatos foram pesquisados,

selecionados, catalogados e analisados. Concluiu-se que as melhores soluções para segurança fim a fim em aplicações *Web Services* são aquelas que aplicam as especificações de Segurança XML.

A resposta obtida para a primeira questão de pesquisa derivou um segundo questionamento: 2ª – Quais soluções de segurança para *Web Services* aplicam as especificações de segurança XML?

Para responder a essa segunda questão de pesquisa, novos trabalhos correlatos foram pesquisados, selecionados, catalogados e analisados. Como resultado, ao se analisar alguns trabalhos pesquisados, detectou-se que o emprego das especificações de segurança XML no nível de mensagem para aplicações da natureza dos *Web Services*, apesar de suprir os requisitos de segurança, promovem também uma degradação no desempenho de tais sistemas, sobretudo, a especificação responsável pela criptografia dos dados: a XML *Encriptyon*. Esta constatação, por sua vez, serviu para formular uma terceira questão, a qual foi útil para se delimitar o escopo do problema tratado neste trabalho: 3ª – Como prover segurança aos *Web Services*, por meio das especificações XML, sem comprometer demasiadamente o desempenho desses sistemas?

Na tentativa de responder a esta terceira questão, novos trabalhos relacionados foram pesquisados, fornecendo as evidências científicas que serviram como diretrizes para a condução deste estudo.

3.1.2 Definição de Termos

Os termos utilizados para realizar a pesquisa *Web* foram extraídos das questões de pesquisa definidas na subseção 3.1.1, da seguinte maneira:

a) Termos de Pesquisa para a primeira questão (Como garantir a segurança fim a fim em aplicações *Web Services*?):
   - Português: “Segurança” e “*Web Services*”,
   - Inglês: “*Security*” and “*Web Services*”.

b) Termos de Pesquisa para a segunda questão (Quais soluções de segurança para *Web Services* aplicam as especificações de segurança XML?):
   - Português: “Segurança” e “*Web Services*” e “Especificações de segurança XML”;
   - Inglês: “*Security*” and “*Web Services*” and “*XML Security Specifications*”.

c) Termos de Pesquisa para a terceira questão (Como prover segurança aos serviços *Web*, por meio das especificações XML, sem comprometer demasiadamente o desempenho desses sistemas?):
   - Português: “Segurança” e “*Web Services*” e “Especificações de segurança XML” e “Desempenho”;
   - Inglês: “*Security*” and “*Web Services*” and “XML *security specifications*” and “*Performance*”.

3.1.3 Critérios de Revisão e Seleção dos Estudos

Após executada a pesquisa *Web*, com base nos termos gerados a partir das questões de pesquisas, deu-se início ao processo de revisão e seleção dos estudos. Durante essa seleção, a avaliação dos títulos e dos resumos (*abstracts*) identificados na busca inicial obedeceu rigorosamente aos critérios de inclusão e exclusão definidos no protocolo de pesquisa. Tais critérios foram determinados com base nas perguntas que nortearam a revisão, da seguinte forma:

a) Critérios de inclusão:
   - pesquisar e analisar as diversas soluções propostas na literatura acadêmica para garantir a segurança em *Web Services*;
   - selecionar apenas os trabalhos que abordem soluções de segurança a nível XML;
   - refinar o processo de pesquisa para trabalhos que abordem não somente o fator segurança, mas também o fator desempenho dos *Web Services* (quando submetidos às especificações de segurança XML);
   - leitura do resumo, em que foram considerados os três primeiros protocolos para inclusão e exclusão;
   - leitura, na íntegra, dos estudos que atenderam a todos os protocolos supracitados para prospecção dos dados relevantes para esta pesquisa.

b) Critérios de exclusão:
   - estudos que não respondam a nenhuma das questões de pesquisa;
   - estudos repetidos ou duplicados.

3.1.4 Análise da Qualidade Metodológica dos Estudos

A qualidade da revisão bibliográfica depende da validade dos estudos nela incluídos. Dessa forma, importa considerar critérios que evitem erros passíveis de comprometer a relevância do estudo em análise. Assim, a fim de se garantir a validade dos trabalhos referenciados neste estudo, foram adotados os seguintes critérios de avaliação:

a) os objetivos e as justificativas para a realização do trabalho em questão são definidos de forma clara?
b) o tipo de estudo está definido sem ambiguidade?
c) o contexto, no qual a pesquisa está inserida, é descrito de forma precisa?
d) o trabalho é devidamente referenciado, ou seja, apresenta trabalhos direta/indiretamente relacionados e, ainda, está baseado em modelos e teorias acadêmicas?
e) os resultados são relatados/descritos de maneira explícita?
f) os resultados obtidos satisfazem os objetivos do estudo ou respondem satisfatoriamente às questões da pesquisa? Existe um método ou um conjunto de métodos descrito para a realização do estudo?

Foram selecionados apenas os trabalhos com respostas afirmativas para os critérios descritos acima.

3.1.5 Fontes de Pesquisa

A seleção das fontes de pesquisa obedeceu aos seguintes critérios:

a) a disponibilidade de consulta dos artigos via *Web*;
b) a disponibilidade de mecanismos de busca por meio de palavras;
c) a relevância das fontes pesquisadas, levando-se em consideração, principalmente, aquelas publicadas nos diversos meios acadêmicos, tais como: congressos, revistas, artigos, dissertações e teses, que tivessem relação com os tópicos identificados na pesquisa.
d) As fontes de pesquisa consideradas foram:
e) Google (http://www.google.com.br);
f) Google Scholar (http://scholar.google.com.br);
g) IEEEXplore Digital Library (http://ieeexplore.ieee.org/Xplore/home.jsp).

3.1.6 Resultados

Como resultado das buscas e após a análise da qualidade metodológica dos estudos, obtiveram-se um total de 7 trabalhos oriundos do Google Scholar e 14 trabalhos oriundos do *IEEE Xplore.* Para o devido registro dos estudos pré-selecionados, foram catalogados os seguintes dados extraídos dos mesmos: (i) Fonte; (ii) Título do trabalho; (iii) Autores; (iv) Local de publicação; (v) Ano de publicação; (vi) Tipo de estudo e; (vii) Foco de pesquisa, na forma da Tabela 1:

Tabela 1 - Catálogo dos Trabalhos Correlatos

Nº	Fonte	Título do trabalho	Autores	Local de publicação	Ano	Tipo	Foco
01	IEEExplore	Quality of Security Service for *Web Services* within SOA	YAMANY, H. F. EL, CAPRETZ, M. A. M.; ALLISON, D. S.	Services - I, 2009 World Conference on	2009	Experimental	Segurança da Informação
02	IEEExplore	Use of Data Mining to Enhance Security for SOA	YAMANY, H. F. EL; CAPRETZ, M. A. M.	Convergence and Hybrid Information Technology, 2008. ICCIT '08. Third International Conference on (Volume:1)	2008	Experimental	Segurança da Informação
03	IEEExplore	"SOAP-based Secure Conversation and Collaboration"	RAHAMAN, M. A.; SCHAAD, A.	In the Proc. of International Conference on *Web Services* (ICWS, 2007)	2007	Experimental	Segurança da Informação
04	IEEExplore	"Design and Implementation of an XMLFirewall"	LOH, Y-S.; YAU, W.C.; WONG, C-T.; HO, W-C.	In the Proc. of International Conference on Computational Intelligence and Security.	2006	Experimental	Segurança da Informação
05	Google Scholar	"A Framework for Enhancing *Web Services* Security"	SIDHARTH, N.; LIU, J.	In the Proc. of 31st Annual International Computer Software and Applications Conference (COMPSAC)	2007	Experimental	Segurança da Informação

Nº	Fonte	Título do trabalho	Autores	Local de publicação	Ano	Tipo	Foco
06	Google Scholar	Comunicação Segura e Confiável para Sistemas Multiagentes Adaptando Especificações XML (OLIVEIRA)	OLIVEIRA, E. J. S.	Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Eletricidade da UFMA.	2006	Experimental	Segurança da Informação
07	Google Scholar	Arquitetura de Segurança em Aplicações Baseadas em *Web Services*	SILVA, R. F.; CUNHA, J. A.	Revista Online HOLOS – ISSN 1807-1600.	2005	Experimental	Segurança da Informação
08	Google Scholar	Avaliação de Desempenho de *Web Services* Seguros – Um Estudo Comparativo	RODRIGUES, D.; BRANCO, K. R. L. J. C.	The Distributed Systems and Concurrent Programming Laboratory (Laboratório de Sistemas Distribuídos e Programação Concorrente – LaSDPC).	2009	Experimental	Segurança da Informação
09	IEEExplore	An adaptive tradeoff model for service performance and security in service-based systems.	YAU, S. S.; YIN, Y.; AN, H. G.	In: ICWS '09: Proceedings of the 2009 IEEE International Conference on *Web Services*, Washington, DC, USA: IEEE Computer Society	2009	Experimental	Segurança da Informação
10	IEEExplore	Towards more secure *Web Services*: Pitfalls of various approaches to XML *Signature* verification process.	KNAP, T.; MLÝNKOVÁ, I.	In: ICWS '09: Proceedings of the 2009 IEEE International Conference on *Web Services*, Washington, DC, USA: IEEE Computer Society	2009	Experimental	Segurança da Informação

Nº	Fonte	Título do trabalho	Autores	Local de publicação	Ano	Tipo	Foco
11	IEEExplore	Performance evaluation and modeling of *Web Services* security.	CHEN, S.; ZIC, J.; TANG, K.; LEVY, D.	IEEE International Conference on *Web Services*, v. 0, p. 431–438	2007	Experimental	Segurança da Informação
12	IEExplore	Identifying opportunities for *Web Services* security performance optimizations.	ENGELEN, R.; ZHANG, W.	In: IEEE Congress on Services - Part I, 2008.	2008	Experimental	Segurança da Informação
13	IEEExplore	An overview and evaluation of *Web Services* security performance optimizations.	ENGELEN, R.; ZHANG, W.	In: IEEE International Conference on *Web Services*, 2008.	2008	Experimental	Segurança da Informação
14	Google Scholar	Performance of web service security	LIU, H.; PALLICKARA, S.; FOX, G.	In: Proceedings of 13th Annual Mardi Gras Conference, Baton Rouge, Louisiana, 2005.	2005	Experimental	Segurança da Informação
15	IEEExplore	A performance evaluation of security mechanisms for *Web Services*.	ALROUH, B.; GHINEA, G.	In: Proceedings of the 2009 Fifth International Conference on Information Assurance and Security - Volume 02, Washington, DC, USA: IEEE Computer Society	2009	Experimental	Segurança da Informação
16	IEEExplore	Server-side streaming processing of ws-security.	GRUSCHKA, N.; JENSEN, M.; IACONO, L.; LUTTENBERGER, N.	IEEE Transactions on Services Computing, v. PP, n. 99, p. 1–14.	2011	Experimental	Segurança da Informação
17	Google Scholar	Comparison of performance of *Web Services*, ws-security, rmi, and rmi-ssl.	JURIC, M. B.; ROZMAN, I.; BRUMEN, B.; COLNARIC, M.; HERICKO, M.	The Journal of Systems and Software, v. 79, p. 689–700.	2006	Experimental	Segurança da Informação

Nº	Fonte	Título do trabalho	Autores	Local de publicação	Ano	Tipo	Foco
18	IEEExplore	Soa and *Web Services*: New technologies, new standards - new attacks.	JENSEN, M.; GRUSCHKA, N.; HERKENHONER, R.; LUTTENBERGER, N.	In: ECOWS '07: Proceedings of the Fifth European Conference on *Web Services*, Washington, DC, USA: IEEE Computer Society.	2007	Experimental	Segurança da Informação
19	IEEExplore	Lye: A high-performance caching soap implementation.	ANDRESEN, D.; SEXTON, D.; DEVARAM, K.; RANGANATH, V.	In: International Conference on Parallel Processing.	2004	Experimental	Segurança da Informação
20	Google Scholar	A performance evaluation of using soap with attachments for e-science.	YING, Y.; HUANG, Y.; WALKER, D. W	In: In Proceedings of the UK e-Science All Hands Meeting.	2005	Experimental	Segurança da Informação
21	IEEExplore	Binary data transfer performance over highlatency networks using web service attachments.	ZHANG, D.; CODDINGTON, P.; WENDELBORN, A.	International Conference on e-Science and Grid Computing, v. 0, p. 261–269.	2007	Experimental	Segurança da Informação

Fonte: Próprio autor (2015).

3.2 RESUMO DOS TRABALHOS CORRELATOS

Nesta subseção, todos os trabalhos que constam na Tabela 1 são resumidos e analisados, sendo elencados na ordem em que foram catalogados:

1. No trabalho "*Quality of Security Service for Web Services within SOA*" **(**YAMANY, 2009), é proposto um metadados para a qualidade do serviço de segurança para SOA, chamado de QoSS (Qualidade de Serviço de Segurança). Esse metadados prevê diferentes níveis para descrever as variações disponíveis dos recursos de autenticação, autorização e privacidade que estão relacionados com a segurança SOA. Relata-se que o metadados (QoSS) se mostra bastante flexível e editável, sendo dividido em quatro níveis básicos de segurança: alta, moderada, baixa e visitante. Tais níveis têm como proposta permitir que variados requisitos de segurança, tanto do lado fornecedor do serviço como do consumidor, sejam satisfeitos. Um serviço *Web* de Qualidade de Serviço de Segurança (QoSS) encapsula esses metadados, a fim de auxiliar o atendimento ao consumidor e provedor de

alcançar um acordo QoSS, que funciona como uma política aplicada para gerenciar as interações entre o prestador de serviços e o consumidor. Como dito anteriormente, o metadados proposto foi encapsulado como um serviço reutilizável, possibilitando ao prestador de serviços publicar sua configuração QoSS, além de ajudar o consumidor a encontrar QoSSs adequados. O consumidor de serviços, por exemplo, é livre para escolher qualquer nível de segurança dos quatro disponíveis (que, por sua vez, estão intrinsecamente relacionados com os três aspectos de segurança SOA: autenticação, autorização e privacidade). No entanto, a seleção do consumidor não deve ter conflito com a política de segurança do prestador de serviços. Este pode requerer que ambos os lados passem por um longo e complicado processo de negociação, visando estabelecer um acordo satisfatório, em termos de requisitos de segurança, para ambos os lados;

2. O trabalho "*Use of Data Mining to Enhance Security for SOA*" **(**YAMANY; CAPRETZ, 2008**)**, por sua vez, propõe um serviço de segurança cujo projeto é baseado em mineração de dados para prever vários ataques *Web*, cujos *Web Services* que recebem solicitações de mensagens SOAP estão sujeitos dentro do ambiente SOA. O modelo de mineração, empregado no referido serviço de segurança, sugere como especificar os tipos de ataque de acordo com os recursos de segurança (como os *tokens* de segurança). Além disso, o modelo usa vários atributos, tais como o tamanho e a análise do tempo de mensagens SOAP, a fim de alcançar previsões confiáveis de que o serviço de segurança depende, para então poder decidir quando permitir ou bloquear a mensagem. A maior experiência dos prestadores de serviços com os ataques *Web* e o elevado número de mensagens SOAP recebidas por esses serviços tornam mais confiável a previsão resultante. Além disso, o modelo de mineração pode ser usado para validar as políticas de segurança, que são geridas usando o *WS-SecurityPolicy*. Esta solução de mineração de dados analisa os recursos sugeridos de segurança, tais como a assinatura e algoritmos de criptografia, até que a melhor configuração com a menor probabilidade de exploração para ataques seja alcançada;
3. Os autores Rahaman e Schaad (2007) atribuíram uma nova seção a uma mensagem SOAP, denominada conta SOAP. Sua função consiste, basicamente, em gravar a estrutura de elementos de uma mensagem SOAP, como o número de elementos de cabeçalho e de elementos assinados. Nesse caso, o principal papel da conta SOAP é proteger as mensagens de receber ataques;
4. Já os autores Loh, Yau, Wong e Ho (2006) sugeriram um *firewall* XML para proteger os *Web Services*, filtrando a entrada de mensagens SOAP. Eles propuseram três diferentes

filtros: filtragem de tamanho da mensagem, análise da sintaxe XML e validação de esquema – segundo os autores, esta abordagem tem alcançado sucesso no bloqueio de alguns dos ataques na *Web*, como cargas de grandes dimensões, cargas recursivas e injeções SQL.

5. Em Sidharth e Liu (2007), apresenta-se uma estrutura para aumentar a segurança dos Serviços *Web* em um ambiente SOA. Esse trabalho discute os ataques na *Web* que podem causar ameaças a cada componente de um ambiente SOA (UDDI, WSDL, SOAP), descrevendo algumas soluções possíveis para proteger cada *Web Serviçe* contra possiveis ameaças que poderiam torná-los vulneráveis. Faz parte dessas possíveis soluções um *framework* chamado *LAPF,* a fim de produzir uma estrutura integral que impediria todos os possíveis ataques;
6. O trabalho "Comunicação Segura e Confiável para Sistemas Multiagentes Adaptando Especificações XML" (OLIVEIRA, 2006) propõe um modelo de comunicação segura e um modelo de entrega de mensagens de forma confiável para sistemas desse gênero. Os dois modelos em questão usam tecnologias derivadas das tecnologias de segurança XML e do padrão RDF (*Resource Description Framework*). Esse modelo de comunicação segura faz adaptações em especificações como: a especificação XML *Signature*, que fornece integridade através de assinatura digital; a XML *Encryption*, que fornece confidenciabilidade através de criptografia; e a XKMS (*XML Key Management Specification*), que fornece suporte ao esquema PKI (*Public Key Infrastructure*). Com o objetivo de fornecer confiabilidade de comunicação, a especificação WS-RM (*WS-Reliable Messaging*) foi adaptada para garantir a entrega das mensagens, permitindo aos agentes trocarem mensagens usando a sintaxe XML;
7. Silva e Cunha (2005), no trabalho "Arquitetura de Segurança em Aplicações Baseadas em *Web Services*", ponderam que, como toda comunicação com *Web Services* é feita através do protocolo SOAP em formato XML, suas mensagens são perfeitamente legíveis e, além disso, SOAP não implementa segurança por questões de manutenção da simplicidade e portabilidade do protocolo. Assim, para que a comunicação seja segura, implementações de segurança devem ser feitas no nível de sistema (nas camadas de rede e de transporte) e nível de aplicação (através de processos e configurações personalizadas e pela aplicação de padrões de segurança em XML). Os autores chegam à conclusão de que, nos padrões baseados em XML, não existe o problema de dependência de plataforma, servidores *Web* ou navegadores, pois XML é um padrão aberto e totalmente compatível com os *Web Services*. Além disso, os padrões de segurança baseados em XML oferecem segurança fim

a fim, fornecendo mecanismos para confidencialidade e integridade do conteúdo das mensagens e capacidade de autenticação e autorização de usuários, o que possibilita uma solução completa para a comunicação segura entre consumidor, intermediários e provedor;

8. Entretanto, no trabalho "Avaliação de Desempenho de *Web Services* Seguros – Um Estudo Comparativo" (RODRIGUES; BRANCO, 2009), cujo objetivo é apresentar um estudo sobre segurança computacional no contexto de arquiteturas orientadas a serviços, compreendendo o desenvolvimento, teste e avaliação de desempenho de políticas de segurança aplicadas em *Web Services*, os autores procuram avaliar a sobrecarga da troca de mensagens com segurança, implementando quatro serviços que executam a mesma operação, ou seja, recebem dois números inteiros e retornam a soma dos mesmos. O diferencial entre os serviços é a política de segurança estabelecida. Dessa forma, relata-se que foi possível a obtenção dos tempos médios de resposta com e sem o uso de políticas de segurança baseadas em especificações XML. Como resultado, observou-se que o fator de maior influência é a criptografia. Torna-se evidente, portanto, que avaliar o impacto e a influência causados por cada algoritmo de criptografia e pelo tamanho das chaves utilizadas pelos mesmos constitui fator importante para a determinação e a obtenção de bons níveis de desempenho em aplicações SOA com segurança. Nesse sentido, os autores do estudo ponderam que isso leva a definições de regras de quais partes devem ser efetivamente e necessariamente criptografadas em uma mensagem SOAP e ainda "qual algoritmo criptográfico é o mais indicado para esta tarefa ou, até mesmo, qual combinação deles (RODRIGUES; BRANCO, 2009);
9. Em Yau et al. (2009), é proposto um modelo de troca adaptativa em sistemas baseados em serviços para melhorar o desempenho e a segurança de sistemas com recursos limitados. Relata-se que tal modelo é capaz de reconfigurar-se a fim de garantir não somente os requisitos de segurança, como também os requisitos de desempenho de aplicações baseadas em arquitetura SOA. Além disso, nesse trabalho, os autores desenvolveram algumas métricas de desempenho e segurança, cuja combinação de ambas, em uma função objetiva com troca de dois fatores de ponderação, define as preferências sobre os fatores de segurança e desempenho com vista a obter o máximo ou, pelo menos, o equilíbrio entre os fatores de segurança e desempenho;
10. Já em Knap e Mlýnková (2009), propõe-se uma abordagem de processamento que possibilite a verificação da integridade de *Web Services*, objetivando a detecção de vulnerabilidades conforme os diversos tipos de ataques a que tais aplicações SOA estão frequentemente expostas. A detecção de tais vulnerabilidades permitiria definir qual

medida de segurança seria necessária para se combater determinado ataque, além de propiciar o devido levantamento das contramedidas a serem utilizadas em cada caso de ataque considerado;

11. Uma avaliação de desempenho para *Web Services* levando em consideração aspectos de segurança é proposta no trabalho de Chen et al. (2007), em que se analisam, principalmente, a variação dos tamanhos de mensagens e também a aplicação de políticas de segurança baseadas em criptografia, assinatura digital e criptografia de uma mensagem assinada, fazendo uso de chaves simétricas e assimétricas. Entretanto, no estudo, os autores não se preocupam em abordar os principais padrões de segurança para *Web Services*, conforme ponderam Rodrigues e Branco (2009) e Silva; Cunha (2005). Estes afirmam que é importante se avaliar as especificações de segurança como principais contramedidas aos ataques proferidos contra esse tipo de aplicação. Também, diferentemente de Rodrigues e Branco (2009), o estudo em questão não considera a variação de desempenho dos diversos algoritmos de criptografia existentes;
12. Oportunidades para otimizar o *WS-Security* são identificadas em Engelen e Zhang, (2008a). Para tanto, os autores analisaram, de forma mais detalhada, a sobrecarga causada pelas operações dessa especificação. Como trabalho futuro (ENGELEN; ZHANG, 2008b), a sobrecarga da especificação *WS-Security* é analisada novamente, com o intuito de se avaliar, desta vez, a aplicação de técnicas de otimização do desempenho da assinatura digital nessa especificação;
13. Desta forma, relata-se que Engelen e Zhang (2008b) compararam diversas técnicas de otimização, indicadas por outros trabalhos relacionados, introduzindo um novo sumário de mensagens baseado em *caching*, objetivando melhorias no desempenho do *WS-Security*. Relata-se também que foi realizada uma avaliação de desempenho levando em conta o impacto dessas otimizações em diferentes tamanhos de mensagens SOAP. Os resultados obtidos apontaram que determinadas combinações das técnicas de otimização aprimoraram, cerca de quatro vezes mais, o desempenho, em termos de velocidade, dos algoritmos empregados no processo de assinatura digital;
14. O trabalho de Liu et al. (2005), por sua vez, analisa o desempenho das operações de segurança. O objetivo desse estudo foi avaliar o desempenho das operações de criptografia, do processamento do SOAP/XML e das implementações do *WS-Security* e do *WS-SecureConversation*. Entretanto, os resultados foram obtidos em uma computação local, ou seja, provedor de serviços e cliente sendo operados na mesma máquina, não se levando em consideração questões importantes, como o tempo que a mensagem gasta

trafegando pela rede e quando as mesmas fazem uso das especificações de segurança. Como uma das conclusões desse material, pode-se comprovar que o tamanho das mensagens SOAP é, realmente, um fator importante no tempo gasto em seu processamento, bem como a complexidade da estrutura da mensagem também deve ser levada em conta. Assim, ao se realizar os processos de criptografia e a assinatura digital em uma mensagem SOAP, por meio da especificação *WS-Security*, acontece um aumento significativo nas operações realizadas e também na complexidade da mensagem (por conta da incorporação de novos elementos XML relacionados à segurança);

15. O desempenho de diversos mecanismos de segurança WSIT (*Web Services Interoperability Technologies*) é analisado em Alrouh e Ghinea (2009). Essa tecnologia é uma proposta de duas conceituadas empresas da tecnologia da Informação: a Sun e a Microsoft, visando à interoperabilidade, de forma aprimorada, entre serviços Java e .Net. Alega-se que os referidos mecanismos (WSIT) são testados em um simples *Web Service*, objetivando avaliá-los com mensagens SOAP, que variam de 1 *byte* a 1 *Mbyte* de tamanho. As conclusões desse estudo comprovaram que mecanismos de segurança da camada de transporte, como o SSL, são significativamente mais eficientes em termos de velocidade do que os mecanismos de segurança fim a fim, ainda mais levando-se em conta o fato de que esse tipo de segurança apresenta o problema de escalabilidade quando mensagens grandes são trocadas, o que não acontece com os mecanismos de segurança da camada de transporte. Apesar disso, o uso de segurança em nível de mensagem é indispensável para a proteção das aplicações *Web Services*, como bem ponderaram Silva e Cunha (2005) e Rodrigues e Branco (2009);
16. Gruschka et al. (2011) propõem um sistema de processamento de *WS-Security* baseado em fluxo, demonstrando como uma mensagem SOAP segura com esse sistema pode ser totalmente processada, incluindo seus elementos de segurança, por fluxo. Relata-se que tal abordagem melhora o desempenho do processamento de documentos XML, se comparada às abordagens mais clássicas, por proporcionar certa economia dos recursos computacionais ao processar tais documentos. De acordo com algumas conclusões obtidas neste trabalho, garantem-se contramedidas mais sólidas para vários tipos de ataques *DoS* (*Denial of Service*), uma vez melhorada a qualidade dos padrões de segurança (como o *WS-Security*);
17. Uma avaliação comparativa sobre o desempenho entre *Web Services* e RMI (*Remote Method Invocation*) foi realizada no estudo de Juric et al. (2006), levando em consideração suas respectivas variações *WSSecurity* e RMI-SSL, que tratam das questões

ligadas ao fator segurança. Relata-se no referido trabalho que tal avaliação foi executada utilizando-se os sistemas operacionais *Windows* e *Linux*. Observou-se, como um dos resultados, que *Web Services* e *WS-Security* apresentam um desempenho inferior ao RMI e RMI-SSL. Isto pode ser explicado, principalmente, pelo aumento considerável do tamanho das mensagens geradas e trocadas pelos *Web Services*, ao fazerem uso da especificação *WS-Security* para prover segurança, o que acarreta um aumento significativo do custo computacional do processamento dessas mensagens. De qualquer forma, levando-se em conta o crescimento exponencial de aplicações do gênero dos *Web Services*, a necessidade de se analisar e aprimorar a relação segurança/desempenho de tais aplicações, por meio de especificações como *WS-Security*, se torna algo imprescindível para a ciência da computação;

18. No trabalho de Jensen et al. (2007), por sua vez, são abordados alguns tipos de ataques mais comumente verificados em *Web Services*, evidenciando-se que estes se encontram expostos, também, a outros tipos de ataques, além daqueles comuns aos protocolos da *Internet*, tais como:
    a) *Coercive Parsing, SOAPAction Spoofing, Attack Obfuscation, BPEL State Deviation e Instantiation Flooding* – estes dois últimos tipos de ataques resultam em uma sobrecarga de CPU (*Central Processing Unit*);
    b) *DoS, Oversized Cryptography, Indirect Flooding e Workflow Engine Hijacking* – também causam sobrecarga de CPU, porém estendem esse efeito à memória;
    c) *XML Injection e WSDL Scanning* – permitem acesso indevido a conteúdo sigiloso;
    d) *Metadata Spoofing* – através desses ataques, podem acontecer escutas ou, até mesmo, a modificação dos dados.

    No estudo em questão, destaca-se também que o fator segurança, geralmente, impacta negativamente o desempenho dos *Web Services*, devido à inexistência de um padrão definido, de fato, para a garantia da segurança nestas aplicações. Também se aponta no estudo que o uso inadequado das especificações de segurança pode comprometer tanto a segurança como o desempenho das aplicações *Web Services*;
19. Por outro lado, o trabalho de Andresen et al. (2004) trata, especificamente, do fator desempenho em *Web Services*, abordando um item de fundamental relevância para a sobrecarga no desempenho dessas aplicações: o processamento das mensagens SOAP no formato XML. Nesse caso, os autores apontam que o maior custo dos processos de codificação e decodificação do XML reside na complexidade estrutural dos elementos, e

não propriamente nos dados que a mensagem transporta. Evidencia-se neste estudo que significativa parte do tempo gasto pelo provedor de serviços é consumida no processo de *marshalling*, que consiste, basicamente, em um processo cuja função é converter a representação, em memória, de um objeto qualquer em dados formatados de forma adequada para armazenamento ou transmissão, ou seja, codificação da mensagem XML. Destaca-se também, no referido estudo, algumas considerações acerca do processamento de mensagens SOAP, que devem ser observadas, sobretudo, em casos nos quais o desenvolvimento ou a integração dos sistemas possui limitações no que se refere ao desempenho. Tais considerações são, sinteticamente, elencadas a seguir:

a) mensagens SOAP resultam em uma considerável sobrecarga em decorrência do tempo elevado para o processamento do XML;
b) 50% do tempo gasto no processamento do XML devem-se à codificação dos dados em XML e à criação da uma conexão HTTP;
c) o processo de codificação de dados binários para o formato XML também acarreta uma sobrecarga no processamento;
d) não existe otimização dos dados quando estes estão formatados em XML;
e) não há balanceamento entre os fatores: velocidade de execução e interoperabilidade;
f) em termos de computação distribuída, SOAP não é tão eficiente, por conta da sua dependência com o XML, quando comparado ao RMI e CORBA (*Common Object Request Broker Architecture*);

20. Já Ying et al. (2005), em outro trabalho que também aborda especificamente o fator desempenho das aplicações que processam dados XML, afirmam que a velocidade de codificação e decodificação, ou seja, a conversão de dados binários para o padrão ASCII (*American Standard Code for Information Interchange*), bem como a conversão no sentido contrário, consiste em um dos principais custos associados ao processamento de dados em XML, sobretudo no que diz respeito à conversão de valores em ponto flutuante e grandes matrizes dimensões;
21. Seguindo a mesma linha de pesquisa dos dois últimos trabalhos discutidos, Zhang et al. (2007) destacam que o alto custo da rede de transmissão, caracterizada pela transferência de dados binários via mensagens SOAP, requer a utilização de algoritmos de codificação binária, como o base64, cuja função é representar dados binários no padrão ASCII – fato que determina alto custo computacional em termos de utilização da CPU e uso de memória, consequentemente degradando o desempenho do sistema. Além do mais,

recorrendo à argumentação exposta anteriormente, os dados convertidos no padrão ASCII são maiores que sua representação binária original, o que gera uma necessidade de maior largura de banda para a transmissão dos dados.

3.3 CONSIDERAÇÕES FINAIS

Como o trabalho desta Dissertação tem foco na promoção da segurança fim a fim dos *Web Services*, proporcionando também uma melhor performance ao desempenho dessas aplicações ao fazerem uso das especificações de segurança XML, considerou-se apropriado condensar, na Tabela 2, os principais aspectos considerados nos estudos analisados que abordam o problema sob a mesma óptica e, ao mesmo tempo, verificar se algum desses trabalhos prevê a aplicação combinada de diversos algoritmos criptográficos como é proposto neste trabalho.

Tabela 2 - Contribuições dos Trabalhos Correlatos

Aspectos Analisados	**Trabalhos Correlatos**							
	Sidharth (2008)	Jensen, (2007)	Engelen (2008a)	Engelen (2008b)	Liu (2005)	Gruschka (2011)	Juric (2006)	Rodrigues; Branco (2009)
Especificações de segurança XML.	✓	—	✓	✓	✓	✓	✓	✓
Algoritmos criptográficos simétricos.	✓	✓	—	—	—	—	—	✓
Algoritmos criptográficos assimétricos.	—	✓	—	✓	—	—	—	—
Tamanho e complexidade da estrutura das mensagens SOAP.	—	✓	—	✓	✓	—	✓	✓
Trafego na Rede.	✓	—	—	—	—	—	—	✓
Diversos algoritmos criptográficos para criptografia de uma mesma mensagem SOAP.	—	—	—	—	—	—	—	—

Fonte: Próprio autor (2015).
Nota: (√) Prevê, (-) Não Prevê.

Como é possível observar, por meio da Tabela 2, dos diversos trabalhos que tratam do problema da degradação de desempenho dos *Web Services* – cuja causa esteja relacionada

direta/indiretamente com a aplicação das especificações de Segurança XML –, nenhum deles considera a hipótese de se aplicar, combinadamente, diversos algoritmos criptográficos. Com exceção do trabalho de Rodrigues e Branco (2009), que, apesar de considerar esta hipótese como uma possível solução, não a leva em conta no desenvolvimento da solução proposta pelos mesmos.

Além do mais, como foi descrito no início deste Capítulo, o principal objetivo desta revisão bibliográfica consistiu em se detectar evidências que servissem como subsídios para a elaboração de uma proposta visando solucionar a problemática abordada neste estudo. Desta forma, com base nos diversos trabalhos correlatos analisados, algumas evidências foram detectadas e definidas como diretrizes para o desenvolvimento da referida solução:

a) das soluções de segurança para *Web Services*, as mais indicadas são aquelas que utilizam as especificações de segurança XML, como XML *Encryption* e XML *Signature*, dentre outras, por proverem segurança fim a fim, entre outras vantagens, como discutido no trabalho de Silva e Cunha (2005). Assim, este trabalho considera a possibilidade de aplicação dessas especificações na solução que propõe;
b) a aplicação de tais especificações de segurança XML pode degradar consideravelmente o desempenho dos *Web Services*, sobretudo no que diz respeito à especificação responsável pela criptografia: XML *Encryption* **(**RODRIGUES; BRANCO, 2009). Desta forma, a criptografia baseada na especificação XML *Encryption* se tornou o aspecto central a ser tratado neste trabalho, quer seja adaptando-a ou, até mesmo, criando uma solução alternativa para ela;
c) a proposta de Yamany, Capretz e Allison (2009), que diz respeito a um metadados chamado de QoSS (Qualidade de Serviço de Segurança) para SOA, dividido em quatro níveis básicos de segurança – alta, moderada, baixa e visitante – juntamente com a hipótese levantada por Rodrigues e Branco (2009) de que a combinação de algoritmos criptográficos pudesse ser uma solução mais eficiente para a se criptografar mensagens XML, subsidiou a ideia de se criar uma solução de segurança para *Web Services* com níveis criptográficos distintos, por meio da aplicação combinada de algoritmos criptográficos, ou seja, um para cada nivel criptográfico, a fim de se alcançar uma melhor performance na execução de *Web Services*, que implementam a segurança em nível de mensagem;
d) também foi considerado, neste trabalho, o requisito de se enviar a chave simétrica, utilizada para criptografia dos dados XML, no cabeçalho da mensagem SOAP, devidamente assinada digitalmente, como foi definido na Arquitetura de Segurança

presente no trabalho de Rodrigues e Branco (2009). Entretanto, com uma distinção: ao invés de se enviar uma única chave, a solução proposta nesse estudo prevê o envio de múltiplas chaves (uma para cada nível criptográfico);

e) outra relevante evidência científica, detectada nesta revisão bibliográfica, diz respeito ao tamanho das mensagens SOAP, bem como à complexidade estrutural dessas mensagens, devido à inserção de novos elementos XML. Pois, como apontam Chen et al. (2007), Liu et al. (2005) e Andresen et al. (2004), este fato causa um impacto degradativo considerável às aplicações *Web Services*, uma vez que demandam maior tempo de processamento dos dados, principalmente dos dados XML. Logo, a solução proposta neste trabalho de Dissertação busca definir poucos elementos XML para estabelecer a segurança fim a fim às mensagens SOAP (basicamente, serão utilizados 3 pares de *tags* XML para definir tanto o procedimento da criptografia como o da assinatura digital).

Seguindo as diretrizes estabelecidas, o presente trabalho propõe o desenvolvimento de uma solução que promova segurança para os *Web Services*, a ser implementada na forma de um *middleware*, denominado de PerSec, capaz de interagir tanto com a aplicação cliente como com o *Web Service*. Este *middleware* fará uso dos algoritmos de criptografia, levando em conta a classificação com base nos fatores de segurança e desempenho, estabelecida também no trabalho de Rodrigues e Branco (2009), para criptografar as mensagens SOAP a depender da confidencialidade dos dados que compõem essas mensagens, ou seja, dados mais confidenciais serão criptografados com algoritmos mais seguros, porém mais custosos computacionalmente, e dados menos confidenciais serão criptografados com algoritmos de menor custo computacional. A estruturação dos dados que compõem a mensagem em níveis de confidencialidade (alta, média e baixa) será definida por meio da utilização de um XML *Schema* criado, especificamente, para esta finalidade.

Dessa forma, espera-se que a solução proposta nesta Dissertação crie regras de quais partes devem ser efetiva e necessariamente criptografadas em uma mensagem SOAP, por meio de uma combinação de algoritmos criptográficos, tendo como objetivo não somente o fator segurança, como também a otimização do fator desempenho dos *Web Services*.

O próximo Capítulo apresenta mais detalhadamente a estrutura da solução proposta.

4 ARQUITETURA E IMPLEMENTAÇÃO DO MIDDLEWARE PERSEC

Este Capítulo tem como objetivo apresentar o *middleware* PerSec, desde seus aspectos mais conceituais até determinados detalhes de seu desenvolvimento. Para tanto, subdivide-se nas seguintes seções:

a) Seção 4.1 – apresenta, em termos conceituais, todo o mecanismo de funcionamento do PerSec que consiste, basicamente, de um *XML Schema* utilizado para estruturar os dados que compõem as mensagens SOAPs em níveis de confidencialidade, a fim de que o *middleware* possa executar suas funções de criptografia, decriptação e autenticação;
b) Seção 4.2 – são apresentados os aspectos da Prototipação do PerSec, abordando-se detalhes das fases de modelagem e implementação do *middleware*;
c) Seção 4.3 – esta última seção, por sua vez, encerra o Capítulo delineando algumas considerações finais acerca de todos aspectos discutidos sobre o PerSec neste Capítulo.

4.1 VISÃO CONCEITUAL DO *MIDDLEWARE* PERSEC

Observa-se que as soluções clássicas de segurança, que visam garantir aspectos de confidencialidade e integridade para *Web Services*, utilizam apenas um algoritmo criptográfico em conjunto com uma chave simétrica para a criptografia dos dados que compõem a mensagem SOAP. No entanto, a análise dos trabalhos correlatos, no Capítulo 3, permitiu identificar que:

a) o uso da especificação responsável pelo processo de criptografia é o que causa maior impacto no desempenho das aplicações baseadas em *Web Services*;
b) determinados algoritmos criptográficos são mais custosos que outros em termos de processamento;
c) a aplicação de diversos algoritmos criptográficos para realizar o procedimento de criptografia, provavelmente, resulta em uma melhora de desempenho nas aplicações *Web Services*.

Com base nessas evidências, considerou-se a possibilidade de se medir o grau de confidencialidade dos dados de uma mensagem SOAP em três níveis – alto, médio e baixo –, aplicando um algoritmo criptográfico para cada nível, conforme sua eficácia em termos de

segurança. Ou seja, o nível alto de confidencialidade será criptografado por meio do algoritmo mais seguro, enquanto o nível médio será criptografado por meio de um algoritmo considerado menos seguro, aplicando-se a mesma lógica ao nível mais baixo de confidencialidade (como cada algoritmo criptográfico apresenta um custo computacional diferente, pondera-se que a aplicação associada desses algoritmos promova uma segurança fim a fim mais equilibrada, em termos de segurança e desempenho, às mensagens SOAP).

Os níveis de confidencialidade serão definidos na mensagem SOAP por meio de um *XML Schema*, denominado de *Levels of Confidentiality*. A Tabela 3 apresenta como este XML *Schema* está estruturado em relação às *tags* e aos algoritmos estipulados para cada nível.

Tabela 3 – Estrutura do XML Schema Levels of Confidentiality

Níveis Confidenciais	**Algoritmos Criptográficos**	***Tags***
Baixo	DES	< LowLevel ></ LowLevel >
Médio	3DES	< MediumLevel ></ MediumLevel >
Alto	AES	< HighLevel ></ HighLevel >

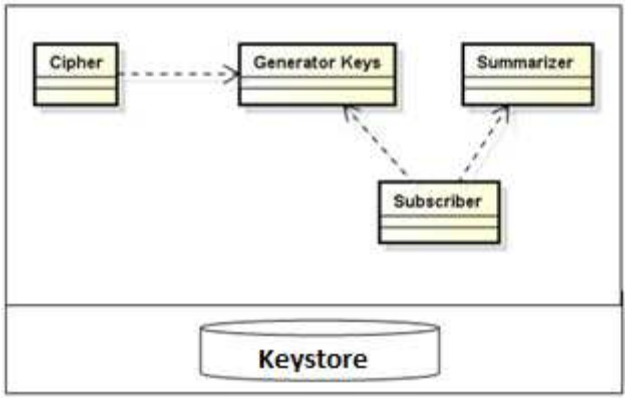
Fonte: Próprio autor (2015).

Este XML *Schema* representa o cerne da solução proposta neste trabalho. Suas *tags*, denominadas de *LowLevel, MediumLevel, e HighLevel*, derivam da abreviação dos termos em português: baixo nível, médio nível e alto nível.

O XML *Schema Levels of Confidentiality* será usado por um *middleware* – ou seja, uma camada de *software* intermediária presente em ambos os lados da conexão, isto é, tanto no cliente como no servidor – para garantir a segurança tanto das aplicações clientes como também dos *Web Services*. Desta maneira, o *middleware* desacoplará a camada de segurança das referidas aplicações, assumindo essa responsabilidade.

Denominado de PerSec (junção das três primeiras letras dos termos Performance e Security), o *middleware* proposto possui a arquitetura ilustrada na Figura 11, adiante.

Figura 11 – Estrutura do Middleware PerSec



Fonte: Próprio autor (2015).

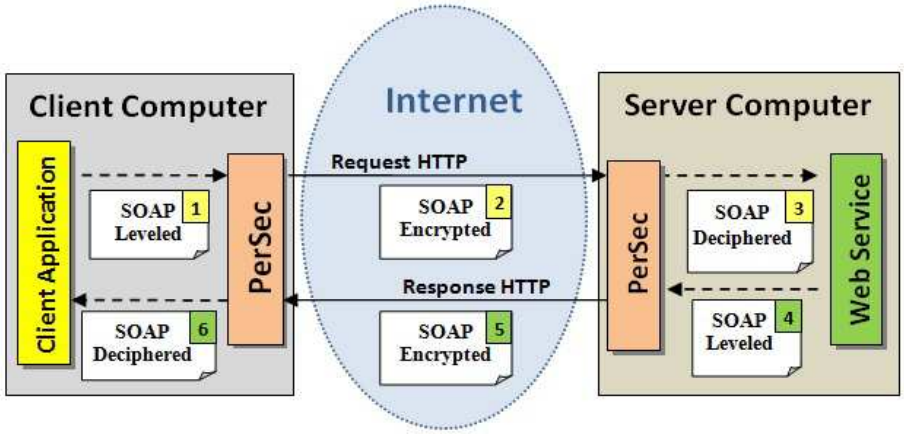
Como é possível observar, por meio da Figura 11, a arquitetura do *middleware* PerSec possui duas camadas: uma camada superior, que contém as classes (*Cipher, Generator Keys, Summarizer e Subscriber*), responsáveis pelo processamento dos serviços de criptografia e assinatura digital; e uma camada inferior, que possui uma base de dados responsável pelo armazenamento das chaves assimétricas, utilizadas pelo PerSec para realizar seus serviços de assinatura e autenticação das mensagens SOAP.

Particularmente, cada componente exerce um papel e determinadas funções nesta arquitetura:

a) *Cipher* – classe responsável pelas tarefas de criptografia e decriptação dos dados;
b) *Generator Keys* – gera as chaves simétricas, conforme os níveis de confidencialidade definidos na mensagem SOAP, por meio do XML *Schema* denominado de *Levels of Confidentiality*;
c) *Summarizer* – gera um resumo da mensagem SOAP para garantir a integridade dos dados que a compõem;
d) *Subscriber* – responsável por assinar tanto o resumo da mensagem como o conjunto de chaves simétricas geradas para o processo de criptografia do corpo da mensagem;
e) *Keystore* – este componente é, basicamente, um repositório de chaves: chaves públicas (no caso do cliente, referentes aos serviços utilizados por este) e chaves privadas (no caso dos *Web Services*).

A Figura 12, a seguir, apresenta uma visão geral do sistema, exemplificando a ação do *middleware* PerSec para promover a garantia da segurança das mensagens SOAP entre uma aplicação cliente e um *Web Service*. O esquema de funcionamento do PerSec apresentado nessa Figura é descrito nos seguintes passos:

Figura 12 – Esquema de Funcionamento do Middleware PerSec

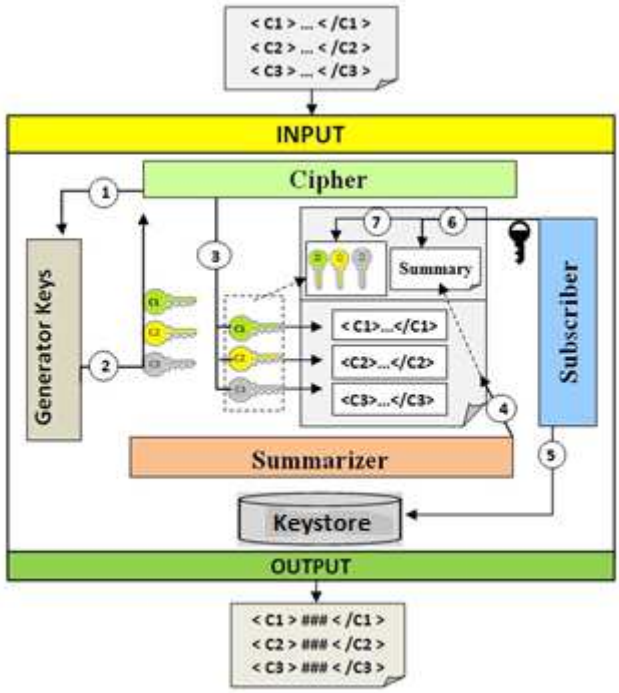


Fonte: Próprio autor (2015).

a) **Passo 1** – a aplicação cliente envia uma mensagem SOAP nivelada, isto é, estruturada em níveis de confidencialidade (por meio do XML *Schema Levels of Confidentiality*), ao *middleware* PerSec para que este aplique os procedimentos de segurança previstos em seus serviços, ou seja, criptografia nivelada e assinatura digital, tanto do resumo da mensagem como das chaves simétricas utilizadas no processo de criptografia;
b) **Passo 2** – a mensagem criptografada pelo *middleware* PerSec é enviada via protocolo HTTP (*request*) ao *Web Service* destinatário;
c) **Passo 3** – o *middleware* PerSec presente na máquina hospedeira do *Web Service* recebe a mensagem SOAP criptografada, realiza o processo de autenticação da mensagem e das chaves e executa o processo de decriptação da mensagem SOAP, transmitindo-a, na sequência, para a aplicação *Web Service*;
d) **Passo 4** – o *Web Service* processa, portanto, a requisição efetuada pela aplicação cliente e envia uma mensagem SOAP nivelada ao *middleware* PerSec para que este aplique as devidas medidas de segurança, ou seja, a criptografia nivelada e a assinatura digital, tanto do resumo da mensagem como também das chaves simétricas utilizadas no processo de criptografia;
e) **Passo 5** – a mensagem SOAP, com o resultado da requisição feita pela aplicação cliente, é enviada para ele de forma criptografada, via protocolo HTTP (*response*);
f) **Passo 6** – o *middleware* PerSec, presente na máquina cliente, ao receber a mensagem SOAP criptografada de resposta do *Web Service*, realiza o processo de autenticação da mensagem e das chaves, bem como o processo de decriptação da mensagem SOAP, transmitindo-a, na sequência, para a aplicação cliente.

Para explicar de forma mais específica os processos de criptografia, decriptação e assinatura digital do PerSec, as Figuras 13 e 14 esquematizam essas funcionalidades, primeiramente, no contexto da multicriptografia das mensagens SOAP e, em seguida, no contexto da decriptação das mensagens multicriptografadas (as *tags*: <*LowLevel*>, <*MediumLevel*> *e* <*HighLevel*>, tanto na Figura 13, como na Figura 14, são representadas, respectivamente, pelas tags: <c1>, <c2> e <c3>).

Figura 13 – Mecanismo de Criptografia do Middleware PerSec



Fonte: Próprio autor (2015).

O processo de criptografia, ilustrado na Figura 12, considerando o papel e a função de cada componente do PerSec, pode ser descrito da seguinte forma:

a) **Etapa 1** – a classe *Cifher* é acionada assim que o *middleware* PerSec recebe a mensagem SOAP estruturada por meio do XML *Schema Levels of Confidentiality*, na aplicação cliente ou pelo *Web Service*, acionando, em seguida, a classe responsável pela geração de chaves (*Generator Keys*);
b) **Etapa 2** – a classe *Generator Keys*, por sua vez, gera as chaves simétricas necessárias para que a classe *Cifher* possa criptografar os dados da mensagem SOAP, de acordo com os níveis de confidencialidade estabelecidos para a mensagem em questão. Ou seja, gera chaves simétricas compatíveis com os algoritmos empregados em cada nível, enviando-as para a classe *Cifher*, que a invocou;

c) **Etapa 3** – após receber como resposta as chaves criadas pela classe *Generator Keys*, a classe *Cifher* realiza o processo de criptografia, aplicando, na mensagem SOAP, o algoritmo criptográfico correspondente a cada nível de confidencialidade definido no XML *Schema Levels of Confidentiality* e, logo após a conclusão do processo de criptografia, insere as chaves simétricas no cabeçalho da mensagem SOAP;

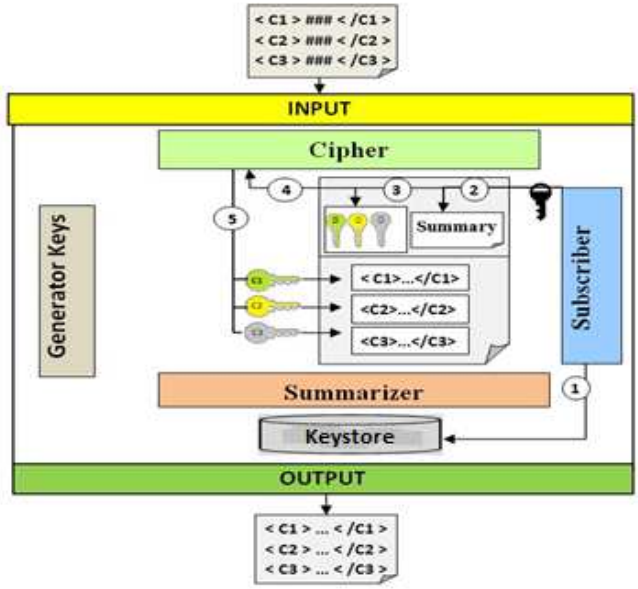
d) **Etapa 4 –** em seguida, a classe *Summarizer* gera o resumo da mensagem SOAP, inserindo-o no cabeçalho dessa mensagem;

e) **Etapa 5 –** a classe *Subscribe*r recupera, na base de dados *Keystore*, a chave Privada (no caso da aplicação *Web Service*) ou a chave pública do *Web Service* (no caso da aplicação cliente);

f) **Etapa 6 –** a classe *Subscriber* assina digitalmente, por meio da chave recuperada no *Keystore*, tanto o resumo da mensagem como as chaves simétricas empregadas na criptografia, para que estas possam ser enviadas juntamente com a mensagem criptografada para o destinatário, a fim de que ele possa realizar a decriptação da mensagem SOAP obtendo seu conteúdo original.

O processo inverso da criptografia – a decriptação da mensagem – obedece ao esquema mostrado na Figura 14.

Figura 14 – Mecanismo de Decriptação do *Middleware* PerSec



Fonte: Próprio autor (2015).

a) **Etapa 1** – ao receber uma mensagem criptografada, a classe *Subscriber* é acionada pelo PerSec para que este efetue os procedimentos de autenticação tanto do resumo

como das chaves simétricas presentes no cabeçalho da mensagem SOAP criptografada, mas, para tanto, a classe *Subscriber* recupera, na base de dados *Keystore*, a chave privada (no caso de a aplicação ser o *Web Service*) ou a chave pública do *Web Service* (no caso de a aplicação ser o cliente).

b) **Etapa 2** – a classe *Subscriber* autentica o resumo da mensagem.
c) **Etapa 3** – em seguida, a classe *Subscriber* autentica as chaves simétricas, enviando-as para a classe *Cipher*.
d) **Etapa 4** – a classe *Cipher*, de posse das chaves devidamente autenticadas, realiza o processo de decifragem dos dados, de acordo com os níveis de confidencialidade, enviando o resultado para a aplicação destinatária.
e) Após a concepção das funcionalidades do *middleware*, explanadas nesta seção, a etapa seguinte do trabalho consistiu em desenvolver um protótipo do sistema em questão, objetivando a realização de um estudo de caso que permitisse analisar a efetividade da técnica de multicriptografia SOAP proposta neste estudo.

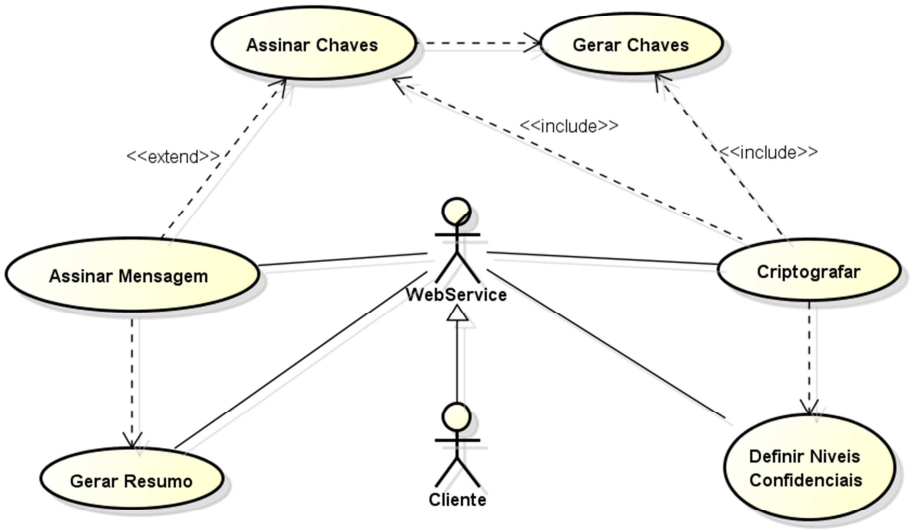
4.2 PROTOTIPAÇÃO DO *MIDDLEWARE* PERSEC

A prototipação do PerSec é apresentada nas próximas duas seções, considerando as fases de modelagem e implementação do *middleware*.

4.2.1 Modelagem

A modelagem do PerSec foi realizada por meio da linguagem UML (*Unified Modeling Language*). Para tanto, utilizaram-se os diagramas de classes (Figura 11 e Figura 19), para fornecer uma visão estrutural do *middleware*, bem como os diagramas de casos de usos e de sequência, para fornecer sua visão comportamental.

Figura 15 - Processo de Criptografia do PerSec



Fonte: Próprio autor (2015).

Os aspectos comportamentais do PerSec podem ser condensados em torno das tarefas principais do *middleware*, ou seja, a criptografia e a decriptação. A Figura 15, acima, expressa os casos de usos previstos para o procedimento da criptografia:

a) **Definir Níveis de Confidencialidade –** as mensagens SOAP compartilhadas entre o *Web Service* e a aplicação cliente deverão ser estruturadas por meio do *XML Schema Levels of Confidentiality*, para que o PerSec possa aplicar os procedimentos de segurança previstos no *middleware*. A Tabela 4, a seguir, expressa esse caso de uso.

Tabela 4 – Caso de Uso Definir Níveis Confidenciais

Característica	Descrição
Ator Primário	*Web Service*/ Cliente
Objetivo	Estruturar os dados da mensagem em níveis confidenciais
Fluxo Principal	1. Define-se o 1ª nível confidencial por meio das *tags*: < *LowLevel* ></ *LowLevel* >; 2. Define-se o 2ª nível confidencial por meio das *tags* <*MediumLevel*> </MediumLevel>; 3. Define-se o 3ª nível confidencial por meio das *tags* <*HighLevel*> </*HighLevel*>;
Fluxo Alternativo	Não se estipula nenhum nível confidencial para a mensagem (nesse caso, julga-se que a mensagem não requer confidencialidade).

Fonte: Próprio autor (2015).

b) **Gerar Chaves** – a Tabela 5 mostra o caso de uso *Gerar Chaves*, no qual o *middleware* verifica os níveis de confidencialidade definidos na mensagem SOAP, gerando uma chave simétrica para cada nível (cada chave gerada é compatível com o algoritmo do nível em questão).

Tabela 5 - Caso de Uso Gerar Chaves

Característica	Descrição
Ator Primário	*Web Service*/ Cliente
Objetivo	Gerar as chaves que serão utilizadas para criptografar os níveis confidenciais definidos na mensagem SOAP.
Fluxo Principal	1. Gera-se uma chave compatível com o algoritmo empregado para criptografar o nível de baixa confidencialidade; 2. Gera-se uma chave compatível com o algoritmo empregado para criptografar o nível média confidencialidade; 3. Gera-se uma chave compatível com o algoritmo empregado para criptografar o nível de alta confidencialidade.
Fluxo Alternativo	Caso nenhum nível seja definido na mensagem, nenhuma chave será gerada.

Fonte: Próprio autor (2015).

c) **Criptografar –** no caso de uso *Criptografar*, Tabela 6, o PerSec criptografa os níveis definidos na mensagem SOAP, utilizando um algoritmo criptográfico específico para cada nível em questão.

Tabela 6 – Caso de Uso Criptografar

Característica	Descrição
Ator Primário	*Web Service*/ Cliente
Objetivo	Realiza a criptografia dos níveis de confidencialidade definidos na mensagem SOAP.
Fluxo Principal	1. Criptografa o nível de baixa confidencialidade com o algoritmo DES; 2. Criptografa o nível média confidencialidade com o algoritmo 3DES; 3. Criptografa o nível de alta confidencialidade com o algoritmo AES;
Fluxo Alternativo	Caso nenhum nível seja definido, nenhum dado será criptografado na mensagem.

Fonte: Próprio autor (2015).

d) Assinar Chaves – neste caso de uso, apresentado na Tabela 7, adiante, as chaves simétricas utilizadas na criptografia dos níveis são inseridas no cabeçalho da mensagem para que possam ser assinadas e enviadas para o destinatário, a fim de que este possa realizar a decriptação dos dados.

Tabela 7 – Caso de Uso Assinar Chaves

Característica	Descrição
Ator Primário	*Web Service*/ Cliente
Objetivo	Assinar as chaves utilizadas para criptografar os níveis confidenciais definidos na mensagem SOAP, enviando-as no cabeçalho da mensagem para que o destinatário possa realizar o processo de decriptação dos dados.
Fluxo Principal	1. Insere as chaves simétricas no cabeçalho da mensagem SOAP; 2. Assina digitalmente as chaves;
Fluxo Alternativo	Caso nenhum nível seja definido na mensagem, não haverá chave para ser assinada.

Fonte: Próprio autor (2015).

e) **Gerar Resumo –** no caso de uso expresso na Tabela 8, o PerSec gera um resumo da mensagem para realizar o processo de assinatura digital da mesma.

Tabela 8 – Caso de Uso Gerar Resumo

Característica	Descrição
Ator Primário	*Web Service*/ Cliente
Objetivo	Gerar um resumo da mensagem para que seja possível realizar o processo de assinatura digital da mensagem em questão.
Fluxo Principal	1. Gera um resumo da mensagem por meio de uma função de *hash*;
Fluxo Alternativo	Caso nenhum nível seja definido na mensagem, ela não será assinada (pois o *middleware* realiza os serviços de criptografia e assinatura digital de forma associada), portanto, não haverá necessidade de se gerar um resumo da mensagem em questão.

Fonte: Próprio autor (2015).

f) **Assinar Mensagem** – neste caso de uso, descrito na Tabela 9, o *middleware* assina o resumo e o insere no cabeçalho da mensagem SOAP.

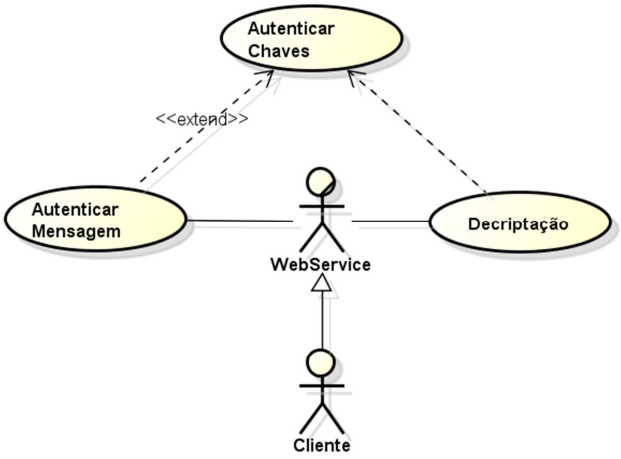
Tabela 9 – Caso de Uso Assinar Mensagem

Característica	Descrição
Ator Primário	*Web Service*/ Cliente
Objetivo	Assinar a mensagem para garantir a autenticidade da mesma.
Fluxo Principal	2. Criptografa por meio de um algoritmo assimétrico o resumo da mensagem; 3. Insere o resumo criptografado no cabeçalho da mensagem.
Fluxo Alternativo	Caso nenhum nível seja definido na mensagem, ela não será assinada, pois, o *middleware* realiza os serviços de criptografia e assinatura digital de forma associada.

Fonte: Próprio autor (2015).

A Figura 16, a seguir, apresenta o diagrama de caso de usos referente ao processo de decriptação das mensagens SOAP.

Figura 16 – Processo de Decriptação do PerSec



Fonte: Próprio autor (2015).

Esses casos de uso podem ser descritos da seguinte forma:

g) **Autenticar Mensagem –** a Tabela 10 descreve o caso de uso *Autenticar Mensagem*, que consiste na verificação de sua integridade, por parte do *middleware* PerSec, realizando a autenticação do resumo assinado pelo emissor da mensagem (cliente ou servidor).

Tabela 10 – Caso de Uso Autenticar Mensagem

Característica	Descrição
Ator Primário	*Web Service*/ Cliente
Objetivo	Verificar a assinatura da mensagem para confirmar a sua autenticidade.
Fluxo Principal	1. Decripta o resumo da mensagem; 2. Utiliza a mesma função de *hash* usada para gerar o resumo recebido; 3. Compara os resumos para verificar a equivalência entre ambos e assim atestar a autenticidade da mensagem recebida.
Fluxo Alternativo	Caso os resumos comparados não sejam equivalentes, a mensagem recebida é considerada como não confiável.

Fonte: Próprio autor (2015).

h) **Autenticar as Chaves –** autentica as chaves simétricas presentes no cabeçalho da mensagem. Na verdade, esse procedimento, descrito na Tabela 11, a seguir, resume-se apenas à decriptação das chaves por meio da chave pública do *Web Service* (caso a decriptação seja realizada pelo cliente), ou pela chave privada do próprio *Web Service* (caso a decriptação seja realizada por este).

Tabela 11 – Caso de Uso Autenticar Chaves

Característica	Descrição
Ator Primário	*Web Service*/ Cliente
Objetivo	Obter as chaves simétricas presentes no cabeçalho da mensagem para decriptar os dados.
Fluxo Principal	1. Decripta as chaves.
Fluxo Alternativo	Caso a mensagem não tenha níveis de confidencialidade definidos, significa que ela não foi criptografada, não havendo, nesse contexto, chaves a serem autenticadas.

Fonte: Próprio autor (2015).

i) **Decriptar Mensagem** – após a autenticação da mensagem e das chaves simétricas, o PerSec realiza a decriptação dos níveis de confidencialidade definidos na mensagem, conforme está descrito na Tabela 12.

Tabela 12 – Caso de Uso Decriptar Mensagem

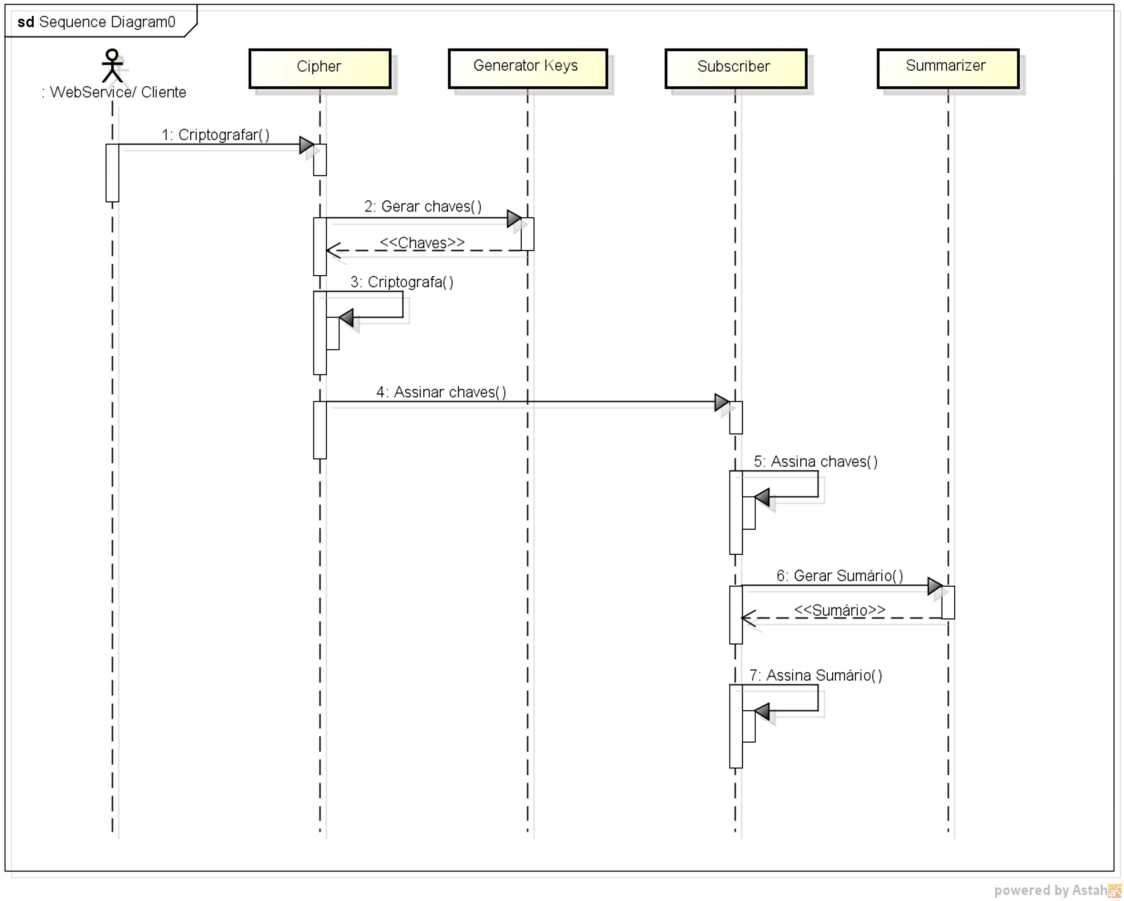
Característica	Descrição
Ator Primário	*Web Service*/ Cliente
Objetivo	Decripta os níveis de confidencialidade da mensagem SOAP para obter a mensagem original.
Fluxo Principal	1. Extrai as chaves simétricas, já autenticadas, do cabeçalho. 2. Utiliza as chaves simétricas para decriptar os dados criptografados, de acordo com os níveis de confidencialidade presentes na mensagem SOAP em questão;
Fluxo Alternativo	Caso a mensagem não tenha níveis de confidencialidade definidos, significa que ela não foi criptografada, não havendo, portanto, dados a serem decriptados.

Fonte: Próprio autor (2015).

Estruturalmente, o *middleware* PerSec pode ser compreendido pelo diagrama de classes apresentado na Figura 11 e na Figura 19 (de forma expandida) que, basicamente, ilustra seu conjunto de classes e os relacionamentos entre essas classes.

Já os diagramas de sequência, representados na Figura 17 e na Figura 18, adiante, têm como objetivo mostrar como as mensagens entre os objetos são trocadas no decorrer do tempo para a realização das operações de criptografia e decriptação, respectivamente.

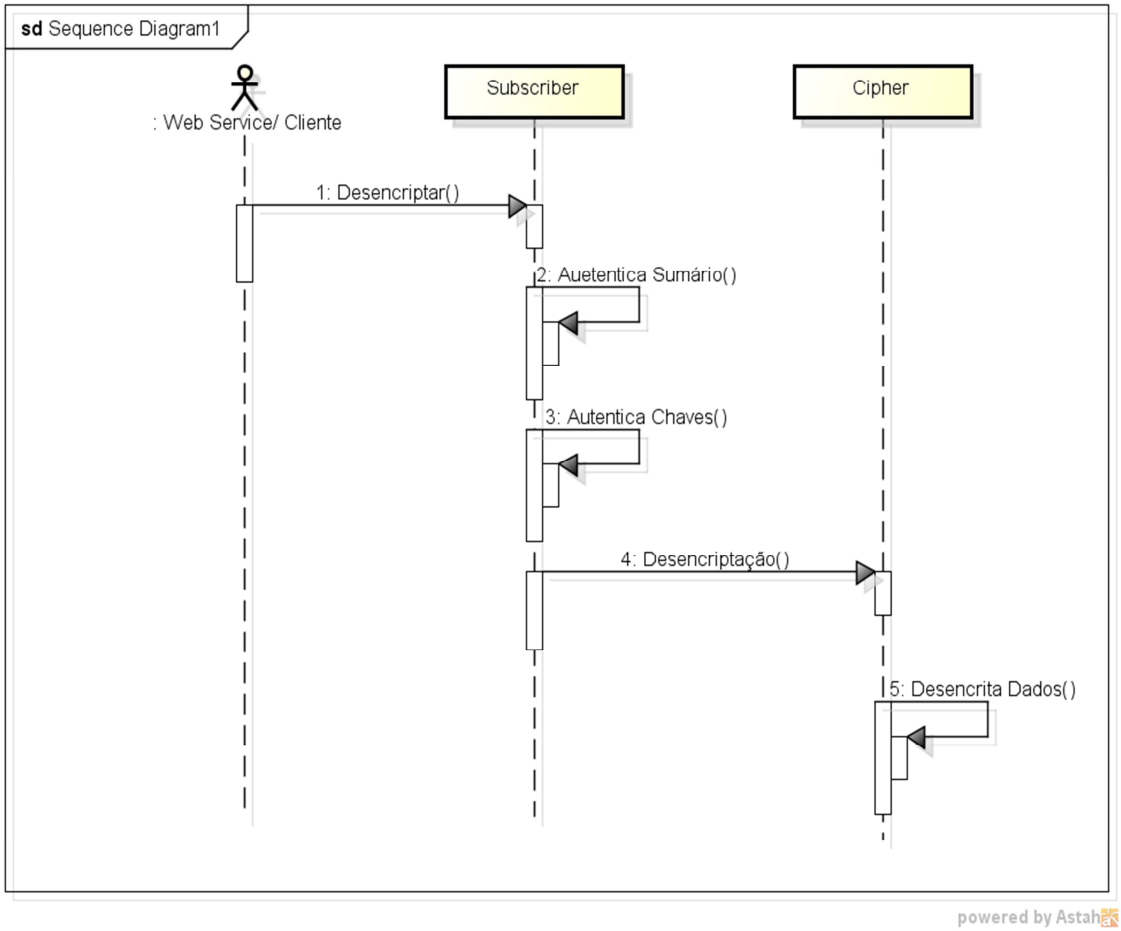
Figura 17 – Diagrama de Sequência do processo de criptografia do PerSec



Fonte: Próprio autor (2015).

A Figura 17 representa o diagrama de sequência para a tarefa de criptografia, que pode ser compreendida da seguinte forma: (i) – A primeira mensagem *criptografar()*, que desencadeia todo o processo, parte do *Web Service* ou da A*plicação Cliente* requisitando ao PerSec a criptografia da mensagem SOAP a ser enviada; (ii) – o objeto *Cipher*, ao receber a mensagem, requisita ao objeto *Generator Keys* que gere as chaves apropriadas para cada nível confidencial definido na mensagem SOAP, por meio do XML *Schema Levels of Confidentiality*; (iii) – o objeto *Generator Keys* gera as chaves solicitadas e as envia para o objeto *Cipher;* (iv) – o *Cipher*, por sua vez, realiza a criptografia dos níveis e, em seguida, requisita a assinatura das chaves ao objeto *Subscriber*; (v) – o objeto *Subscriber* assina as chaves e as insere no cabeçalho da mensagem SOAP; (vi) – o objeto *Subscriber* requisita o sumário do objeto *Summarizer*, que envia o resumo da mensagem para o objeto solicitante; (vii) – o objeto *Subscriber* assina a mensagem e a insere no seu cabeçalho, finalizando o processo de criptografia e assinatura digital do PerSec.

Figura 18 – Diagrama de Sequência do processo de decriptação do PerSec



Fonte: Próprio autor (2015).

A Figura 18, por sua vez, representa o digrama de sequência para a tarefa de decriptação. Esse diagrama pode ser interpretado da seguinte maneira: (i) – o cliente ou o *Web Service* envia uma mensagem (*Desencriptar()*) para o PerSec, que é captada e processada pelo objeto *Subscriber;* (ii) – o objeto *Subscriber*, então, realiza a autenticação do *Sumário* e, posteriormente, faz também a autenticação das **chaves** contidas no cabeçalho da mensagem SOAP; (iii) – o objeto *Subscriber* envia uma mensagem (*Desencriptação()*), contendo como parâmetros as chaves simétricas pertencentes aos níveis, para a classe *Cipher*, que realiza a decriptação (*Desencripta Dados()*) dos níveis de confidencialidade da mensagem.

A próxima seção apresenta os principais detalhes técnicos envolvidos na fase de implementação do *middleware* PerSec, abordando aspectos como linguagem de programação, ferramentas e APIs empregadas, bem como explicações gerais sobre a implementação do *middleware.*

4.2.2 Implementação do PerSec

O PerSec[1] foi implementado por meio da linguagem de programação *Java*, empregando-se, para tanto, uma diversidade de ferramentas e APIs, tais como:

j) a plataforma *Java SE Security*, que inclui uma diversidade de APIs, ferramentas, mecanismos, protocolos e implementações de algoritmos, que são comumente utilizados para prover o fator segurança às aplicações *Java*. As APIs de segurança *Java* dessa plataforma abrangem vários aspectos de segurança, como criptografia, infraestrutura de chave pública, comunicação segura, autenticação e controle de acesso (ORACLE, 2015). Esta API foi utilizada para implementar todas as funcionalidades de segurança do *middleware* PerSec, como criptografia, assinatura digital e geração de chaves simétricas e assimétricas;

k) API JAX-WS: API *Java* para XML *Web Services* (JAX-WS) fornece uma maneira padrão de desenvolver *Web Services* interoperáveis e móveis. O modelo de programação JAX-WS fornece um recurso manipulador de aplicativos que possibilita manipular uma mensagem em um fluxo de entrada ou de saída. Este recurso é chamado de *handlers*. É possível inclui-los no ambiente de tempo de execução JAX-WS para executar o processamento adicional de mensagens de pedido e resposta. É possível, também, utilizar *handlers* para uma variedade de fins, como capturar e registrar em *log* as informações e incluir a segurança ou outras informações em uma mensagem (IBM, 2015). No desenvolvimento do projeto PerSec, implementaram-se duas classes *handlers*: a *ClientHandler* e a *ServerHandler*, que, apesar de não integrarem o *middleware*, são responsáveis por instanciar os objetos das classes que compõem o *middleware* PerSec, acionando seus métodos, tanto nas aplicações cliente como no próprio *Web Service*;

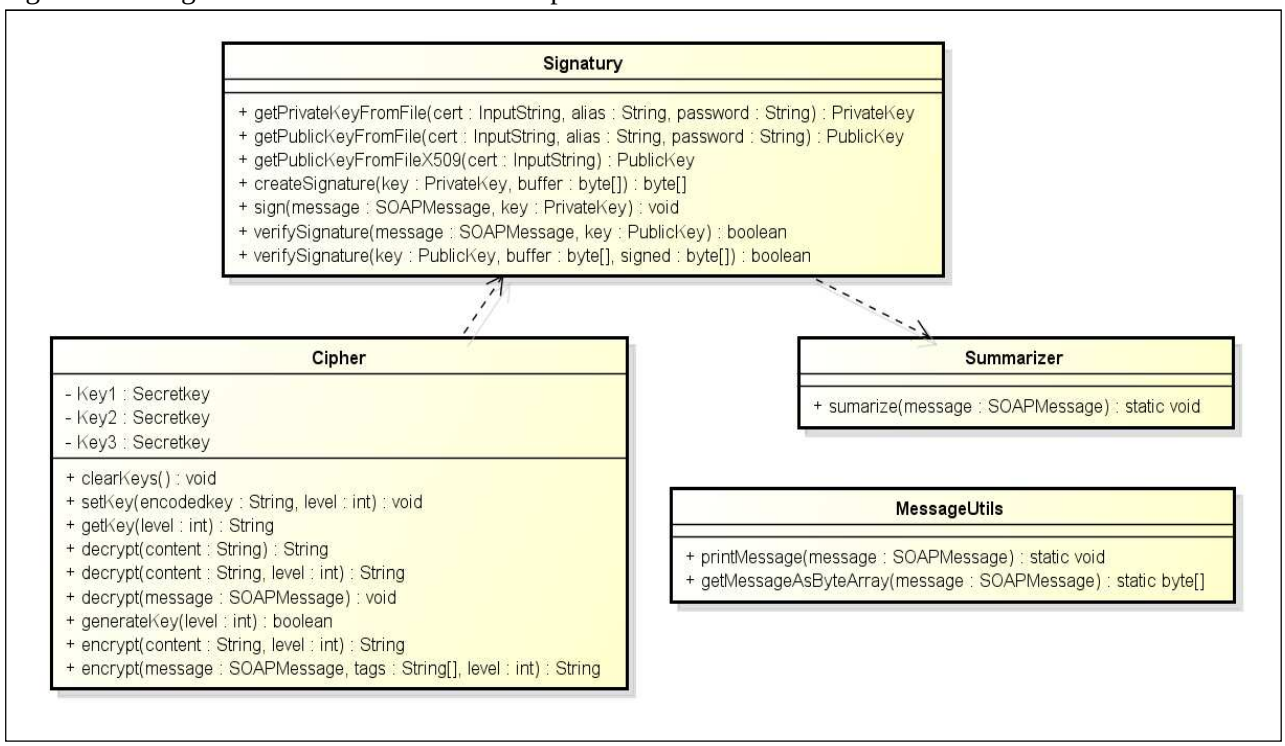
l) A API JDOM (JDOM, 2015), uma biblioteca *Java* de acesso e manipulação de documentos XML.

A Figura 19 apresenta o diagrama de classes do *middleware* PerSec expandido, após implementação, contendo uma nova classe chamada de *MessageUtils* (utilizada para exibir a estrutura das mensagens SOAP processadas pelo PerSec) e também as classes *Cipher*, *Signature e Summarizer*, com seus atributos e métodos, concebidas na arquitetura do *middleware*.

[1] A documentação javadoc do *middleware* PerSec está presente no Apêndice A.

Além dessas classes, o componente *Generator Keys*, também previsto na arquitetura do *middleware*, foi substituído pelos métodos *GenerateKey* e *getKey* na classe *Cipher*, uma vez que verificou-se que essa funcionalidade já era oferecida pela *API Java SE Security* (empregada neste projeto para implementar algumas das funcionalidades do *middleware* PerSec), seguindo, assim, um princípio básico da engenharia de *software*, que é a reutilização de componentes prontos.

Figura 19 - Diagrama de Classes do PerSec Expandido



Fonte: Próprio autor (2015).

Duas classes auxiliares são responsáveis por acionar os serviços do *middleware* PerSec, tanto no lado do cliente – a *ClientHandler* – como no lado do *Web Service*: a *ServerHandler*. Essas duas classes foram desenvolvidas com base na API *Java* para XML *Web Services* (JAX-WS), chamada de *handlers*. Estes, como já descrito anteriormente, são recursos manipuladores de aplicativos, que possibilitam manipular uma mensagem em um fluxo de entrada ou de saída.

O acionamento dos serviços de segurança do *middleware* PerSec nas mensagens SOAP, de pedido e resposta entre o cliente e o *Web Service*, acontece por meio de *handlers* programados tanto do lado cliente, como do servidor, como mostram as Tabelas 13 e 14.

De um modo geral, as classes *ClientHandler* e *ServerHandler* se comportam de forma similar. Ambas verificam se a mensagem SOAP a ser processada representa uma mensagem de saída

ou uma mensagem de entrada, por meio do método *handleMessage()* (linha 25 na Tabela 13 e linha 28 na Tabela 14).

Tabela 13 - Implementação da Classe do ClientHandler

Linhas	Código
01	`package banking.client;`
02	
03	`import java.text.SimpleDateFormat;`
04	`import java.util.Date;`
05	`import java.util.GregorianCalendar;`
06	`import java.util.Set;`
07	
08	`import javax.xml.namespace.QName;`
09	`import javax.xml.soap.SOAPMessage;`
10	`import javax.xml.ws.handler.MessageContext;`
11	`import javax.xml.ws.handler.soap.SOAPHandler;`
12	`import javax.xml.ws.handler.soap.SOAPMessageContext;`
13	
14	`import persec.*;`
15	
16	`public class ClientHandler implements SOAPHandler<SOAPMessageContext> {`
17	
18	`public void close(MessageContext context) {`
19	`}`
20	
21	`public boolean handleFault(SOAPMessageContext context) {`
22	`return false;`
23	`}`
24	
25	`public boolean handleMessage(SOAPMessageContext context) {`
26	`try {`
27	`Boolean isMessageOut =`
28	`(Boolean)context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);`
29	
30	`SOAPMessage message = context.getMessage();`
31	
32	`Cipher cipher = new Cipher();`
33	
34	`if (isMessageOut) {`
35	`GregorianCalendar start = new GregorianCalendar();`
36	
37	`cipher.encrypt(message, new String[] {"agency"}, 1);`
38	`cipher.encrypt(message, new String[] {"number"}, 2);`
39	`cipher.encrypt(message, new String[] {"password"}, 3);`
40	
41	`Summarizer.summarize(message);`
42	
43	`*Signature*.sign(message,`
44	`*Signature*.getPrivateKeyFromFile(getClass().getResourceAsStream("/banking/certs`
45	`/banking-cert.jks"), "banking.keys", "123456"));`
46	
47	`GregorianCalendar finish = new GregorianCalendar();`
48	
49	`MessageUtils.printMessage(message);`
50	`System.out.println("*SIGNATURE* + SUMMARY + ENCRYPT [" + new`
51	`SimpleDateFormat("dd/MM/yyyy HH:mm:ss:SSS").format(new`
52	`Date(start.getTimeInMillis())) + " - " +`
53	`new SimpleDateFormat("dd/MM/yyyy`
54	`HH:mm:ss:SSS").format(new Date(finish.getTimeInMillis())) + "] --> " +`

Continua

Linhas	Código
55	(finish.getTimeInMillis() -
56	start.getTimeInMillis()) + " ms");
57	}
58	else {
59	System.out.println("\nSOAP MESSAGE RESPONSE WEB
60	SERVICE");
61	
62	GregorianCalendar start = new GregorianCalendar();
63	
64	if (!*Signature*.verify*Signature*(message,
65	*Signature*.getPublicKeyFromFileX509(AccountClient.class.getResourceAsStream("
66	/banking/certs/banking-cert.x509"))))
67	return false;
68	
69	if (!Summarizer.verifySummary(message))
70	return false;
71	
72	cipher.decrypt(message);
73	
74	GregorianCalendar finish = new GregorianCalendar();
75	
76	MessageUtils.printMessage(message);
77	
78	System.out.println("*SIGNATURE*/SUMMARY VALIDATION +
79	DECRYPT [" + new SimpleDateFormat("dd/MM/yyyy HH:mm:ss:SSS").format(new
80	Date(start.getTimeInMillis())) + " - " +
81	new SimpleDateFormat("dd/MM/yyyy
82	HH:mm:ss:SSS").format(new Date(finish.getTimeInMillis())) + "] --> " +
83	(finish.getTimeInMillis() -
84	start.getTimeInMillis()) + " ms");
85	}
86	
87	return true;
88	}
89	catch(Exception e) {
90	e.printStackTrace();
91	return false;
92	}
93	}
94	
95	public Set<QName> getHeaders() {
96	return null;
97	}
98	
100	}
101	

Fonte: Próprio autor (2015).

Tabela 14 - Implementação da Classe do ServerHandler

Linhas	Código
01	package banking.services.handler;
02	
03	import java.text.SimpleDateFormat;
04	import java.util.Date;
05	import java.util.GregorianCalendar;
06	import java.util.Set;
07	
08	import javax.xml.namespace.QName;
09	import javax.xml.soap.SOAPMessage;
10	import javax.xml.ws.handler.MessageContext;
11	import javax.xml.ws.handler.soap.SOAPHandler;
12	import javax.xml.ws.handler.soap.SOAPMessageContext;
13	
14	import banking.client.AccountClient;
15	import persec.Cipher;
16	import persec.*Signature*;
17	import persec.Summarizer;
18	
19	public class ServerHandler implements SOAPHandler<SOAPMessageContext> {
20	
21	public void close(MessageContext context) {
22	}
23	
24	public boolean handleFault(SOAPMessageContext context) {
25	return false;
26	}
27	
28	public boolean handleMessage(SOAPMessageContext context) {
29	try {
30	Boolean isMessageOut =
31	(Boolean)context.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);
32	
33	SOAPMessage message = context.getMessage();
34	
35	Cipher cipher = new Cipher();
36	
37	if (!isMessageOut) {
38	GregorianCalendar start = new GregorianCalendar();
39	
40	if (!*Signature*.verify*Signature*(message,
41	*Signature*.getPublicKeyFromFileX509(AccountClient.class.getResourceAsStream
42	("/banking/certs/banking-cert.x509"))))
43	return false;
44	
45	if (!Summarizer.verifySummary(message))
46	return false;
47	
48	cipher.decrypt(message);
49	
50	GregorianCalendar finish = new GregorianCalendar();
51	
52	System.out.println("\n*SIGNATURE*/SUMMARY VALIDATION +
53	DECRYPT [" + new SimpleDateFormat("dd/MM/yyyy HH:mm:ss:SSS").format(new
54	Date(start.getTimeInMillis())) + " - " +
55	new SimpleDateFormat("dd/MM/yyyy
56	HH:mm:ss:SSS").format(new Date(finish.getTimeInMillis())) + "] --> " +
57	
58	
59	(finish.getTimeInMillis() –

Continua

Linhas	Código
60	start.getTimeInMillis()) + " ms");
61	}
62	else {
63	GregorianCalendar start = new GregorianCalendar();
64	
65	cipher.encrypt(message, new String[] {"date"}, 1);
66	cipher.encrypt(message, new String[] {"balance"}, 2);
67	cipher.encrypt(message, new String[] {"name"}, 3);
68	
69	Summarizer.summarize(message);
70	
71	*Signature*.sign(message,
72	*Signature*.getPrivateKeyFromFile(getClass().getResourceAsStream
73	("/banking/certs/banking-cert.jks"), "banking.keys", "123456"));
74	
75	
76	GregorianCalendar finish = new GregorianCalendar();
77	
78	System.out.println("\n*SIGNATURE* + SUMMARY + ENCRYPT ["
79	+ new SimpleDateFormat("dd/MM/yyyy HH:mm:ss:SSS").format(new
80	Date(start.getTimeInMillis())) + " - " +
81	new SimpleDateFormat("dd/MM/yyyy
82	HH:mm:ss:SSS").format(new Date(finish.getTimeInMillis())) + "] --> " +
83	(finish.getTimeInMillis() -
84	start.getTimeInMillis()) + " ms");
85	}
86	
87	return true;
88	}
89	catch(Exception e) {
90	e.printStackTrace();
91	return false;
92	}
93	}
94	
95	public Set<QName> getHeaders() {
96	return null;
97	}
98	
99	}
100	

Fonte: Próprio autor (2015).

Caso seja uma mensagem de entrada (*input*), as classes *handlers* da aplicação cliente e do *Web Service* acionam o método *encripty()* da classe *Cypher* do PerSec (linhas 37 a 39 na Tabela 13 e 63 a 65 na Tabela 14) para criptografar os dados da mensagem SOAP, de acordo com os níveis de confidencialidade definidos na mensagem. Em seguida, as classes *handlers* em questão acionam o método *sumarize()* (linhas 41 da Tabela 13 e 67 da Tabela 14) pertencente à classe *Summarizer* do *middleware* PerSec para gerar um sumário da mensagem SOAP, a fim de que o processo de assinatura digital possa ser realizado. Por fim, as classes *ClientHandler* e *ServerHandler* acionam o método *sign()* (linhas 43 na Tabela 13 e 69 na

Tabela 14) pertencente à classe *Signature* do PerSec para aplicar a assinatura digital à mensagem SOAP tratada.

Por outro lado, as classes *handlers*, ao detectarem que a mensagem SOAP é uma mensagem de saída (*output*), por meio do mesmo método *handleMessage()*, já descrito anteriormente, reconhecem a autenticidade da mensagem SOAP, acionando, para tanto, o método *verifySummary(),* pertencente à classe *Summarizer* do PerSec (linha 64 na Tabela 13 e linha 45 na Tabela 14). Em seguida, as classes *ClientHandlers* ou *ServerHandlers* acionam o método *decrypt()* (linha 72 da Tabela 13 e linha 48 da Tabela 14) da classe *Cipher* do *middleware* PerSec para decriptar os dados da mensagem SOAP, conforme seus níveis de confidencialidade.

Vale ressaltar que os processos de criptografia e assinatura digital não foram implementados por meio das especificações XML *Signature* e *Encryption*, apesar de a revisão bibliográfica realizada neste trabalho ter apontado essas especificações como sendo as mais apropriadas para se promover a segurança em nível de mensagem nas aplicações *Web Services*. Os motivos de não utilizar tais especificações residem nos seguintes fatos:

a) as referidas especificações são definidas por meio de uma série de elementos XML inseridos nas mensagens SOAPs, tais como *EncriptedData*, *EncryptionMethod*, *KeyInfo*, *CipherData*, *CipherValue*, dentre outras (como foi visto no Capítulo 2), aumentando consideravelmente o tamanho das mensagens SOAP. Entretanto, a proposta da solução apresentada neste trabalho consiste em se utilizar apenas três pares de *tags* XML *(<LowLevel>, <MediumLevel> e <HighLevel>*) para estruturar as mensagens SOAP e, assim, garantir a confidencialidade e a autenticidade dessas mensagens (por meio do processamento das mesmas pelo *middleware* PerSec);
b) o desenvolvedor precisa lidar com uma série de questões que devem ser definidas ao utilizar as especificações de segurança XML, como, por exemplo, determinar os algoritmos criptográficos a serem utilizados; indicar como as chaves, simétrica e assimétrica, podem ser recuperadas, dentre outros aspectos (que são, totalmente, comtempladas pelo PerSec, liberando o desenvolvedor de lidar com essas questões);
c) a especificação XML *Encryption* não prevê a utilização de vários algoritmos criptográficos, nem tampouco várias chaves. Portanto, adaptar essa especificação para implementar o processo de multicriptografia não seria uma tarefa trivial, talvez, inviável.

Por conta dessas considerações, optou-se por implementar o *middleware* PerSec como uma solução alternativa às soluções que utilizam as especificações de segurança XML: *Signature* e *Encryption*, como o WS-Security.

4.3 CONSIDERAÇÕES FINAIS

Neste Capítulo, o *middleware* PerSec foi descrito considerando as fases de concepção e implementação da ferramenta. Sua aplicação foi realizada levando em conta todos os aspectos estruturais e comportamentais do *middleware*, que foram definidos na fase de concepção do sistema, tratados na Seção 4.1 deste Capítulo. Entretanto, algumas alterações no projeto conceitual foram realizadas na fase de implementação do protótipo, tais como:

a) a classe *Generator Keys* não foi implementada, pois foi constatado que a *API Java Security* já disponibilizava o recurso de geração de chaves simétricas e assimétricas, por conta disso, foram elaborados métodos na classe *Cypher*, baseados na API *Java Security*, para cumprir esta finalidade. Portanto, houve uma mudança na arquitetura do PerSec, porém, em futuras implementações, considerando-se outras tecnologias, a classe *Generator Keys* poderá vir a ser desenvolvida, visando-se alcançar os benefícios provindos de dois princípios da engenharia de *software*, que são o desacoplamento e o reúso;

b) foi criada uma classe auxiliar – *MessageUtils* – não prevista na concepção do *middleware*, apenas para exibir as mensagens SOAPs tratadas pelo PerSec;

c) foram criadas também duas classes *handlers – ClienteHandler e ServerHandler* **–** que não incorporam o *middleware*, mas que trabalham em conjunto com o mesmo, acionando os seus serviços ao manipularem as mensagens SOAP de entrada e saída em tempo de execução.

d) Além das alterações realizadas, algumas limitações também foram encontradas:

e) o compartilhamento da chave pública entre o *Web Service* e as aplicações clientes não seguiu o padrão de distribuição por meio de uma Autoridade Certificadora (CA), devido ao alto custo desse tipo de serviço, tornando-o inviável para um projeto acadêmico como este. Por conta disso, as chaves foram compartilhadas por meio de um diretório comum às aplicações em questão;

f) o PerSec, nessa primeira versão, opera somente com aplicações *Web Services* desenvolvidas em *Java*, no entanto, objetiva-se uma versão futura mais universal do *middleware*, ou seja, que trabalhe com aplicações desenvolvidas em qualquer

linguagem de programação, a fim de se garantir uma propriedade SOA fundamental, que é a interoperabilidade entre aplicações, não importando em qual linguagem tenham sido implementadas.

Apesar das limitações encontradas no desenvolvimento do PerSec, a execução da fase de testes demonstrou que o protótipo do *middleware* atendeu, de forma adequada, os requisitos funcionais do seu projeto. Portanto, foi considerado apto para executar a avaliação experimental prevista neste trabalho, que será descrita no próximo Capítulo.

5 AVALIAÇÃO DO MIDDLEWARE PERSEC

Este Capítulo descreve a avaliação experimental do *middleware* PerSec realizada nesta Dissertação, que teve como objetivos:

a) comprovar a hipótese considerada no presente estudo, que consiste na utilização de diversos algoritmos criptográficos, ao invés de apenas um algoritmo (prática usual) para realizar a criptografia das mensagens SOAP e, dessa forma, aprimorar o desempenho de *Web Services* que implementam a segurança fim a fim, ou seja, a segurança em nível de mensagem;
b) avaliar o comportamento do *middleware* PerSec ao garantir a segurança tanto dos *Web Services* como das aplicações clientes desses serviços.

Para tanto, a avaliação experimental foi projetada e executada com base nos principais cenários de uso do referido *middleware*, uma vez que tais cenários possibilitam que a mensagem SOAP seja criptografada com um ou mais algoritmos, por meio dos seus níveis de confidencialidade. Assim, este Capítulo está estruturado da seguinte forma:

a) **Seção 5.1 – Domínio da Aplicação:** nesta seção, são descritas as características da aplicação que será utilizada para executar os experimentos, a fim de se alcançar os objetivos (supracitados) definidos para esta etapa do trabalho;
b) **Seção 5.2 – Configuração do Ambiente de Testes:** o objetivo desta seção é descrever todos os elementos de *hardware* e *software* envolvidos no contexto desta avaliação;
c) **Seção 5.3 – Planejamento dos Experimentos**: nesta etapa, definem-se quais fatores serão considerados nesta avaliação do *middleware* PerSec, bem como o tipo de planejamento a ser adotado para a realização dos experimentos propostos;
d) **Seção 5.4 – Análise dos Resultados:** esta seção, basicamente, consiste na observação analítica dos resultados obtidos com a realização dos experimentos, buscando-se, dessa forma, validar estatisticamente a avaliação experimental proposta;
e) **Seção 5.5 – Considerações Finais:** nesta última seção, considerações finais acerca dos experimentos realizados são explanadas.

5.1 DOMÍNIO DA APLICAÇÃO

Para realizar a avaliação experimental proposta, desenvolveu-se um *Web Service* de

Aplicação Bancária denominado *BankingApp*[2], que oferta aos seus clientes o serviço de saldo bancário.

As requisições de saldo realizadas pelas aplicações clientes necessitam informar os seguintes dados como parâmetros para o *Web Service*: número da conta, agência e senha. Esses dados são criptografados pelo *middleware* PerSec, conforme os níveis de confidencialidade definidos para cada experimento (como será apresentado numa seção posterior), assinados digitalmente e enviados para o *Web Service* via protocolo SOAP/HTTP.

Em contrapartida, o *Web Service* responde à solicitação do cliente enviando uma mensagem SOAP, contendo os seus dados bancários. Vale ressaltar que tais dados também são criptografados pelo PerSec (presente no lado do servidor) nos níveis de confidencialidade definidos em cada experimento, assim como na mensagem SOAP de requisição. Em seguida, a mensagem é assinada digitalmente pelo *middleware*, para que assim possa ser encaminhada para a aplicação cliente.

Dessa forma, o *middleware* PerSec garante os principais aspectos de segurança às mensagens SOAP, compartilhadas entre o *Web Service* e as aplicações clientes, ou seja, garante os aspectos de confidencialidade, integridade e autenticidade.

5.2 CONFIGURAÇÃO DO AMBIENTE DE TESTES

Em experimentos computacionais, é importante listar todos os componentes de *hardware* e de *software* neles envolvidos, a fim de se identificar qual é a relevância desses elementos nos resultados auferidos.

Como o objetivo principal desta avaliação experimental reside em testar o desempenho, no que se refere ao tempo de execução, de *Web Services,* cujas mensagens SOAP sejam criptografadas por meio de diversos algoritmos, resolveu-se executar os experimentos, previstos nesta avaliação, como uma aplicação local, ou seja, com a aplicação cliente e o *Web Service* sendo executados na mesma máquina. A Tabela 15 descreve a configuração básica da referida máquina.

[2] A documentação javadoc da Aplicação *BankingApp* encontra-se disponível no *Apêndice B.*

Tabela 15 - Configuração de Hardware da Máquina de Teste

Componente	Quantidade	Características
Processador	1	Intel Core I5-3317U, 1.7GHz.
Memória RAM	1	6GB
HDD	1	500GB+24 GB (SSD).

Fonte: Próprio autor (2015).

Em relação aos componentes de *software*, a aplicação *BankingApp* foi implementada por meio da linguagem de programação *Java*, sendo utilizados também outros componentes de *software*, tais como:

a) *Apache Tomcat* 8.0 (APACHE, 2015b) – servidor de aplicação para implementações em *Java* como *Servlets* e páginas JSP (*JavaServer Pages*);
b) *PostGreSQL* 9.4 (POSTGREESQL, 2014) – gerenciador de banco de dados *open source*.
c) *Windows* 8.1(MICROSOFT, 2015) – sistema operacional utilizado para executar os experimentos.

5.3 PLANEJAMENTO DOS EXPERIMENTOS

Esta etapa tem como objetivo definir os fatores que serão examinados, a fim de se analisar a influência de cada um deles no desempenho do sistema computacional avaliado (JAIN, 1991). Isto implica determinar a quantidade de dados coletados, a quantidade de replicações dos experimentos (fator crucial para se validar estatisticamente os resultados obtidos) e, ainda, a possível interação entre os fatores considerados.

Importa destacar alguns termos envolvidos no contexto do projeto e análise de experimentos (JAIN, 1991), tais como: (i) Variável de Resposta – corresponde à saída de dados esperada de um determinado experimento (nesse caso, a medida de desempenho do sistema); (ii) Fatores – representam as variáveis que influenciam a variável de resposta; (iii) Níveis – significa o domínio de valores que podem ser atribuídos a um determinado fator; (iv) Interação – diz respeito à possibilidade de dependência entre os fatores considerados no experimento.

A avaliação de desempenho utilizada neste estudo de caso está baseada na metodologia do planejamento simples, que consiste em se determinar uma configuração inicial, fixando cada um dos fatores e variando os demais. Desta maneira, torna-se possível analisar a influência de cada fator no desempenho do sistema em questão. Este tipo de planejamento é muito empregado e de fácil implementação, contudo, apresenta a desvantagem

de não permitir avaliar a interação entre os fatores (JAIN, 1991). Foi escolhido pelo fato de que o estudo de caso desenvolvido neste trabalho analisa apenas um fator isolado, uma vez que o *middleware* PerSec sempre realiza as operações de criptografia e de assinatura digital de forma associada, não havendo, portanto, a necessidade de se avaliar a interação entre estes dois fatores de maneira isolada, pois tal possibilidade não é considerada pelo *middleware*. Além do mais, outros fatores indiretos, como número de clientes, não são considerados, pois, o que se pretende verificar com esta avaliação experimental, principalmente, é o desempenho dos *Web Services* ao realizarem a criptografia dos dados das mensagens SOAP com diversos algoritmos criptográficos. Pretende-se, também, avaliar o desempenho do *middleware* PerSec **de uma maneira geral**, ao prestar o seu serviço de segurança fim a fim.

A Tabela 16 apresenta o único fator considerado nesta avaliação, assim como seus respectivos níveis. O fator em questão, como descrito no parágrafo anterior, diz respeito à criptografia e à assinatura digital. Os níveis, por sua vez, assumem como valores os algoritmos simétricos utilizados para criptografar os níveis de confidencialidade das mensagens SOAP associados ao algoritmo de chave pública, empregado para assinar digitalmente tais mensagens.

Tabela 16 - Fatores e níveis dos experimentos

Fatores	Níveis
Criptografia e assinatura digital	Um algoritmo simétrico e um de chave pública
	Dois algoritmos simétricos e um de chave pública
	Três algoritmos simétricos e um de chave pública

Fonte: Próprio autor (2015).

A Tabela 17, adiante, apresenta os experimentos executados, com base no fator e níveis estipulados.

Tabela 17 – Experimentos Realizados

Experimentos	Tipo
1	Dados criptografados apenas no nível de baixa confidencialidade (*LowLevel*)
2	Dados criptografados apenas no nível médio de confidencialidade (*MediumLevel*)
3	Dados criptografados apenas no nível alto de confidencialidade (*HighLevel*)
4	Dados criptografados nos níveis de baixa e média confidencialidade
5	Dados criptografados nos níveis de baixa e alta confidencialidade
6	Dados criptografados nos níveis de média e alta confidencialidade
7	Dados criptografados nos três níveis (baixa, média e alta) de confidencialidade

Fonte: Próprio autor (2015).

Algumas considerações importantes sobre os experimentos realizados podem ser elencadas da seguinte maneira:

a) a criptografia realizada no primeiro nível de confidencialidade é efetuada por meio do algoritmo DES (*Data Encryption Standard*);
b) a criptografia realizada no segundo nível de confidencialidade é efetuada por meio do algoritmo 3DES (*Triple Data Encryption Standard*);
c) a criptografia realizada no terceiro nível de confidencialidade é efetuada por meio do algoritmo AES (*Advanced Encryption Standard*);
d) a assinatura digital é efetuada por meio do algoritmo RSA/SHA-1 (WEIS, 2006);
e) a variável de resposta considerada para os experimentos realizados neste estudo de caso será o RTT (*Round Trip Time*), que compreende o tempo gasto no processo de requisição/resposta entre a aplicação cliente e o *Web Service*. Esse tipo de variável de resposta é bastante utilizado para se avaliar o desempenho de *Web Services* (JURIC et al., 2006; ALROUH; GHINEA, 2009). Entretanto, consideraram-se apenas as fatias de tempo de processamento do processo de criptografia (realizado pelo PerSec instalado na aplicação cliente) e do processo de decriptação (realizado pelo PerSec instalado no servidor do *Web Service*), a fim de se analisar esses processos, considerando apenas uma mensagem SOAP (no caso, optou-se pela mensagem *Request*);
f) para efeitos de abreviação, convencionou-se que os níveis: baixo (*LowLevel*), médio (*MediumLevel*) e alto (*HighLevel*) serão referenciados, no restante deste Capítulo, respectivamente, como: nível 1, nível 2 e nível 3.

O primeiro, segundo e terceiro experimentos serviram para representar o sistema clássico de criptografia das mensagens SOAP, ou seja, por meio de apenas um algoritmo criptográfico. Já o quarto, quinto, sexto e sétimo representam o sistema de criptografia proposto neste trabalho, no qual se empregam dois ou mais algoritmos criptográficos para realizar o procedimento de criptografia das mensagens SOAP. Tais experimentos possibilitaram uma análise comparativa que serviu como prova de conceito para a hipótese adotada neste estudo, como foi descrito no início deste Capítulo.

Decidiu-se replicar 30 vezes cada experimento, objetivando-se, assim, alcançar uma validação estatística para cada um, pois Jain (1991) considera essa quantidade de replicações suficiente para se garantir intervalos de confiança consideravelmente baixos, possibilitando, desta maneira, uma comparação satisfatória dos resultados.

A Tabela 18 apresenta o corpo da mensagem SOAP *Request* (mensagem de requisição da aplicação cliente) considerada, neste estudo de caso, para a análise do processo de criptografia (realizado no PerSec instalado na máquina cliente) e também para análise do processo de decriptação (realizado no PerSec instalado no servidor do *Web Service*).

Tabela 18 – Mensagens SOAP Request e Response

Linha	Mensagem SOAP *Request*
1	<S:Body>
2	<ns2:getBalance xmlns:ns2="http://services.banking/">
3	<agency>1880</agency>
4	<number> 197963</number>
5	<password>12345</password>
6	</ns2:getBalance>
7	</S:Body>

Fonte: Próprio autor (2015).

Os elementos nivelados confidencialmente, para cada experimento, na mensagem *Request*, foram: <*agency*>, <*number*> e <*password*>. Para cada experimento, os referidos elementos foram estruturados em níveis de confidencialidade de uma determinada maneira. Os fragmentos da Tabela 18, elencados por experimento, demonstram tal estruturação:

- Experimento 1:

<nível 1><agency>1880</agency></nível 1>
<nível 1><number> 197963</number><nível 1>
<nível 1><password>12345</password><nível 1>

- Experimento 2:

<nível 2><agency>1880</agency></nível 2>
<nível 2><number> 197963</number><nível 2>
<nível 2><password>12345</password><nível 2>

- Experimento 3:

<nível 3><agency>1880</agency></nível 3>
<nível 3><number> 197963</number><nível 3>
<nível 3><password>12345</password><nível 3>

- Experimento 4:

<nível 1><agency>1880</agency></nível 1>
<nível 2><number> 197963</number><nível 2>
<nível 2><password>12345</password><nível 2>

- Experimento 5:

<nível 1><agency>1880</agency></nível 1>
<nível 3><number> 197963</number><nível 3>
<nível 3><password>12345</password><nível 3>

- Experimento 6:

<nível 2><agency>1880</agency></nível 2>
<nível 3><number> 197963</number><nível 3>
<nível 3><password>12345</password><nível 3>

- Experimento 7:

<nível 1><agency>1880</agency></nível 1>
<nível 2><number> 197963</number><nível 2>
<nível 3><password>12345</password><nível 3>

Uma observação importante (perceptível pela estruturação do corpo da mensagem *Request*) em relação ao nivelamento confidencial das mensagens SOAP reside no fato de que essa tarefa depende exclusivamente do julgamento do programador em determinar quais elementos de uma mensagem SOAP são confidenciais e qual é o nível da confidencialidade de cada um desses elementos. Portanto, este aspecto na aplicação do *middleware* PerSec para se garantir a segurança fim a fim das mensagens SOAP é de crucial relevância para a performance do sistema.

5.4 ANÁLISE DOS RESULTADOS

Os dados obtidos com a execução dos experimentos foram analisados considerando a média dos resultados obtidos nas 30 execuções para cada experimento. Vale ressaltar que, a fim de se validar, estatisticamente, esses resultados, efetuaram-se alguns cálculos estatísticos, como o desvio padrão e o intervalo de confiança de 95% para cada experimento realizado, assim como propõe Jain (1991).

Como um dos objetivos desta avaliação experimental consiste em analisar o comportamento geral do *middleware* PerSec ao prestar seus serviços de segurança, considerou-se, inicialmente, tanto as RTTs dos processo de criptografia como também o processo de decriptação dos níveis de confidencialidade – vale ressaltar que deve ser computado nessas RTTs o processo de assinatura digital, na etapa da criptografia e do reconhecimento da assinatura e das chaves simétricas, no processo de decriptação, uma vez que o PerSec sempre executa esses procedimentos de forma associada ao prestar seus serviços de segurança. Assim, a Tabela 19 e a Figura 20 apresentam os resultados obtidos para a etapa

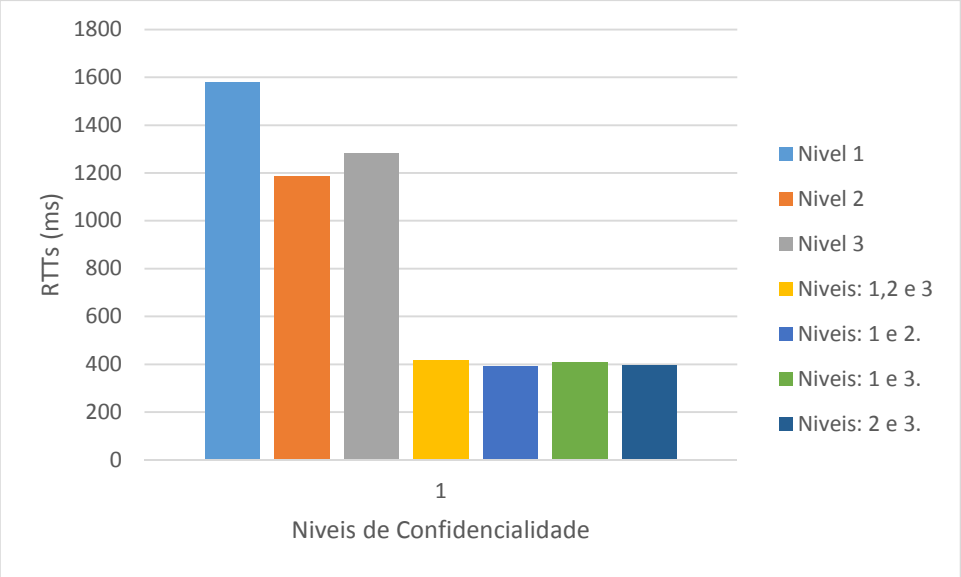
de criptografia, enquanto a Tabela 12 e a Figura 21 apresentam os resultados obtidos para a etapa da decriptação.

Tabela 19 – RTT da Criptografia por Níveis de Confidencialidade

Experimento	Média dos RTT	Desvio padrão	Amplitude do Intervalo de confiança
Nível 1	1579,83	152,84	113,84
Nível 2	1186,23	35,92	26,20
Nível 3	1283,60	94,79	70,60
Níveis: 1 e 2	392,86	20,72	15,42
Níveis: 1 e 3	406,30	23,33	17,08
Níveis: 2 e 3	394,93	18,47	13,76
Níveis: 1, 2 e 3	415,93	14,46	10,76

Fonte: Próprio autor (2015).

Figura 20 – RTT da Criptografia por Níveis de Confidencialidade



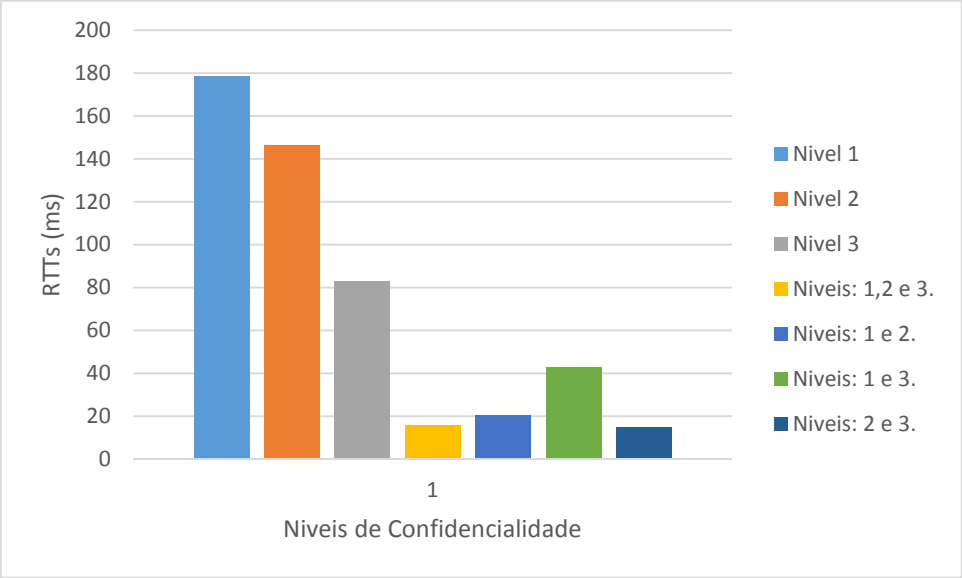
Fonte: Próprio autor (2015).

Tabela 21 - RTT da Decriptação por Níveis de Confidencialidade

Experimento	Média dos RTT	Desvio padrão	Amplitude do Intervalo de confiança
Nível 1	178,36	33,85	25,20
Nível 2	146,40	19,51	14,52
Nível 3	82,80	12,28	9,14
Níveis: 1 e 2	42,66	7,08	5,26
Níveis: 1 e 3	20,26	7,38	5,50
Níveis: 2 e 3	16,03	3,05	2,26
Níveis: 1, 2 e 3	14,80	2,68	2,00

Fonte: Próprio autor (2015).

Figura 21 - RTTs da Decriptação por Níveis de Confidencialidade



Fonte: Próprio autor (2015).

Ao se analisar os dados contidos nas Tabelas 19 e 20, bem como os gráficos expressos nas Figuras 20 e 21, percebe-se claramente que os experimentos, tanto no que diz respeito ao processo de criptografia quanto no processo de decriptação, utilizando dois ou mais níveis de confidencialidade, apresentam melhores resultados do que aqueles experimentos que empregaram apenas um nível de confidencialidade.

Em relação aos experimentos que empregaram apenas um nível de confidencialidade, no que se refere ao processo de criptografia, nota-se que o nível 2 se mostrou ligeiramente mais eficaz que o nível 3 e praticamente 25% mais eficaz que o nível 1, que, por sua vez, demonstrou ser menos eficaz, em torno de 17%, do que o nível 3. Já em relação ao processo de decriptação, considerando também os experimentos que empregaram apenas um nível de confidencialidade, o nível 3 foi cerca de 43% mais eficaz do que o nível 2 e praticamente acima de 53% mais eficiente que o nível 1, que, por sua vez, apresentou um desempenho, em torno de 18%, pior do que o nível 2.

Por outro lado, os experimentos que aplicaram um ou mais níveis de confidencialidade apresentaram resultados mais eficazes do que os experimentos que utilizaram apenas um nível de confidencialidade. A Tabela 18 e a Figura 20 mostram que os experimentos que aplicaram, respectivamente, níveis 1 e 2; níveis 1 e 3; níveis 2 e 3 e níveis 1, 2 e 3 efetuaram mais ou menos a operação de criptografia na faixa dos 400 ms, sendo mais eficazes cerca de 71% em relação ao experimento de nível 3; 66% em relação ao experimento nível 2 e 75% em relação ao experimento de nível 1.

Já no que diz respeito ao processo de decriptação empregando vários níveis de confidencialidade, observou-se que o experimento de níveis 1, 2 e 3 foi ligeiramente superior aos experimentos de níveis 1 e 3 e de níveis 2 e 3, respectivamente. Por outro lado, mostrou ser cerca de 65% mais eficaz do que o experimento de níveis 1 e 2. Entretanto, apesar de o experimento de níveis 1 e 2 ter obtido o pior resultado no quesito decriptação, considerando vários níveis de confidencialidade, ele ainda se mostrou bem superior, em torno de 48%, ao resultado obtido pelo melhor experimento aplicando apenas um nível de confidencialidade, mais especificamente, o experimento de nível 3.

Todas as comparações discutidas, entre os experimentos realizados neste estudo de caso, podem ser observadas na Tabela 21, que apresenta uma análise comparativa de desempenho nível por nível/níveis, levando em conta os processos de criptografia e decriptação. Vale ressaltar que essa análise comparativa diz respeito à eficácia dos algoritmos em termos de custo de processamento, e não em termos de robustez do fator segurança de cada algoritmo.

Pelos resultados mostrados nas Figuras 20 e 21, pode-se observar também que o *middleware* PerSec apresentou um custo computacional bem mais acentuado ao realizar o procedimento de criptografia do que ao executar o procedimento de decriptação dos níveis de confidencialidade – o que, aparentemente, pode significar que o processo de criptografia em si, de fato, é computacionalmente mais custoso do que o seu processo inverso (a decriptação), uma vez que o processo de assinatura digital (realizado de forma associada pelo PerSec ao multicriptografar as mensagens SOAP), segundo Rodrigues e Branco (2009), não causa impacto significativo no desempenho dos *Web Services*.

Além disso, analisando-se o desvio padrão e o intervalo de confiança expressos nas Tabelas 19 e 20, verificou-se, de início, que, em relação ao desvio padrão (uma medida de dispersão usada com a média que, basicamente, mede a variabilidade dos valores em volta da média), o coeficiente de variação desses dados nas referidas tabelas, em sua grande maioria, não ultrapassa a barreira dos 20%, indicando baixa dispersão com referência à média dos RTTs analisados, comprovando a homogeneidade da amostra. Já em relação aos intervalos de confiança (que podem ser compreendidos como os intervalos estimados em que a média de um parâmetro de uma amostra tem uma dada probabilidade de ocorrer), não se verificou a sobreposição desses dados, comprovando que os resultados obtidos são estatisticamente diferentes. Essas duas observações estatísticas são premissas importantes para a validade do estudo em questão.

5.5 CONSIDERAÇÕES FINAIS

A prática desta avaliação experimental do PerSec serviu para confirmar a hipótese sustentada neste trabalho: a técnica de utilizar diversos algoritmos criptográficos é mais eficiente do que a técnica convencional de utilizar apenas um algoritmo para realizar o processo de criptografia em mensagens SOAP. Entretanto, não se observou uma diferença significativa ao se empregar dois ou três níveis no processo de criptografia. Por outro lado, o processo de decriptação apresentou resultados bem mais significativos (em termos de custo computacional), principalmente ao se utilizar três níveis de confidencialidade, ou seja, três algoritmos criptográficos.

Uma observação importante a ser considerada é a forma como a mensagem SOAP é estruturada por meio do XML *Schema Levels of Confidentiality* para definir os níveis de confidencialidade dos dados da mensagem. Como essa tarefa depende intrinsecamente do julgamento do programador, talvez dados não confidenciais possam ser definidos como confidenciais, acarretando impacto degradativo no desempenho do sistema.

Nos experimentos realizados nesta avaliação, por exemplo, na mensagem *Request*, o nome do cliente foi definido com um nível 1 de confidencialidade, devido à mensagem SOAP apresentar poucos dados e também ao objetivo de testar o desempenho do sistema. Em uma mensagem SOAP, contudo, com uma grande quantidade de dados, talvez um dado como um nome de uma pessoa não tenha tanta relevância, em termos de confidencialidade, para necessitar ser criptografado. Logo, a tarefa de estruturar os dados de uma mensagem SOAP em níveis de confidencialidade, para que assim possam ser multicriptografados, é um aspecto de crucial importância para o desempenho das aplicações *Web Services* que empregarem o *middleware* PerSec com o intuito de garantir a segurança fim a fim das suas mensagens SOAP.

Como foi descrito anteriormente neste Capítulo, mais especificamente na seção 5.2 (Configuração do Ambiente de Testes), em experimentos computacionais, é importante listar todos os componentes de *hardware* e de *software* envolvidos neles, a fim de se identificar qual é a relevância desses elementos nos resultados auferidos. Neste caso, ressalta-se a importância de se investigar mais acuradamente, em experimentos futuros, a influência de um processador com múltiplos núcleos (como o utilizado nos experimentos deste estudo) nos resultados da multicriptografia SOAP, uma vez que os objetos gerados pelo processo de criptografia poderiam ser armazenados em núcleos diferentes, influenciando, dessa forma, no desempenho do *middleware* em relação aos experimentos realizados.

O próximo Capítulo delineia as últimas considerações finais a respeito do trabalho, apresentando também algumas perspectivas em termos de trabalhos futuros, bem como a produção científica auferida.

6 CONCLUSÃO

A realização deste trabalho permitiu constatar que aplicações baseadas em *Web Services* necessitam garantir, além da segurança ponto a ponto (por meio de tecnologias, como SSL/TLS), também a segurança fim a fim, ou seja, em nível de mensagem (geralmente, por meio das especificações de segurança XML *Encryption* – responsável pelo processo de criptografia – e a XML *Signature* – responsável pelo processo de assinatura digital). Entretanto, o processo de criptografia, utilizando-se a especificação XML *Encryption* (levando em conta o custo computacional dos algoritmos criptográficos e o tamanho das chaves simétricas que, geralmente, são empregadas) ocasiona impacto degradativo considerável no desempenho dos *Web Services*.

Assim, a revisão bibliográfica realizada nesta Dissertação indicou que os algoritmos criptográficos e o tamanho das chaves empregadas podem ser considerados como uma das principais causas da degradação de desempenho provocada pelo processo de criptografia. Outro indício importante apontado pela referida revisão bibliográfica diz respeito à hipótese levantada por Rodriguez e Branco (2009), ao ponderarem que a utilização de diversos algoritmos criptográficos, talvez, resultasse em um aprimoramento do desempenho dos *Web Services*, ao invés de se utilizar apenas um algoritmo criptográfico para garantir a confidencialidade dos dados de uma mensagem SOAP.

Tal hipótese foi testada, neste trabalho, por meio da implementação de um *middleware*, denominado PerSec (junção das três primeiras letras dos termos *performance* e *security*), cujo mecanismo de funcionamento se baseia em estruturar os dados das mensagens SOAP em níveis de confidencialidade, por meio de um *XML Schema*, criptografando cada nível através de um algoritmo criptográfico distinto. Portanto, a solução proposta, neste estudo, pode ser considerada uma alternativa às soluções mais clássicas, que se baseiam nas especificações de segurança XML, já que utilizam apenas um algoritmo criptográfico para codificar os dados confidenciais das mensagens SOAP.

O estudo experimental, descrito no Capítulo 5, permitiu inferir que a hipótese adotada para o desenvolvimento deste trabalho é verdadeira, isto é, o uso de multicriptografia em *Web Services* é mais eficiente do que a prática usual de usar apenas um algoritmo criptográfico, pois os resultados obtidos demostraram que, ao se utilizar dois, ou três, algoritmos para se criptografar a mesma mensagem SOAP, o tempo de execução do processo de criptografia e também da decriptação é reduzido significativamente, o que resulta num melhor desempenho

para as aplicações (*Web Services* e aplicações clientes) que implementam esse tipo de segurança.

Além disso, alguns outros benefícios proporcionados pelo PerSec, identificados durante a realização deste trabalho, podem ser elencados da seguinte forma:

a) uma configuração predefinida de segurança, que seja automaticamente reconhecida do outro lado da conexão, pode suprimir a etapa de estabelecimento de uma política de segurança entre as partes envolvidas na comunicação, o que resultaria, mesmo que indiretamente, em um aumento de performance por eliminar uma etapa de implementação da segurança em *Web Services*;
b) o XML *Schema Levels of Confidentiality* pode reduzir a complexidade da estrutura das mensagens SOAP, uma vez que não é necessária a inclusão de informações sobre qual o tipo de algoritmo criptográfico utilizado, ou ainda, dados sobre o modo de recuperação de chaves para realizar a decriptação;
c) o desacoplamento da segurança das aplicações para um *middleware*, como o PerSec, reduz a complexidade de desenvolvimento dos *Web Services* que primem pela segurança, além de garantir a interoperabilidade entre essas aplicações, mesmo que desenvolvidas em linguagens e plataformas diferentes (apesar de este requisito não ter sido atendido nesta primeira versão do PerSec).
d) Entretanto, algumas observações devem ser consideradas:
e) o estudo de caso proposto foi executado utilizando apenas uma máquina, ou seja, com a aplicação cliente e o *Web Service* executados no mesmo computador. Isto porque o objetivo principal, neste caso, foi medir o tempo de processamento da criptografia e da decriptação dos níveis de confidencialidade, por isso, foi desprezado o tempo de tráfego de rede, que também seria calculado caso experimentos realizados fossem executados em um ambiente mais próximo do real, ou seja, com o *Web Service* e a aplicação cliente executados em máquinas distintas, conectadas por uma rede e com diferentes configurações;
f) a gama de dados da mensagem SOAP, considerada na avaliação da proposta, que requisitou o tratamento de segurança do *middleware* PerSec foi pequena. Neste caso, seria apropriado executar a multicriptografia SOAP em mensagens maiores em termos de dados confidenciais, com a finalidade de se verificar o comportamento do PerSec, quanto aos resultados permanecerem persistentes, ou não, com os resultados obtidos no estudo experimental realizado;

g) o *middleware* PerSec não demonstrou diferenças consideráveis em termos de tempo de execução para o processamento da criptografia envolvendo vários níveis de confidencialidade, por outro lado, na decriptação da mensagem estruturada pelos três níveis, obtiveram-se melhores resultados em relação às outras possibilidades de múltipla nivelação confidencial;
h) a relativização envolvida no julgamento do desenvolvedor em determinar quais dados são mais confidenciais do que os outros também é um aspecto relevante a ser considerado, pois a nivelação confidencial dos dados pode impactar, tanto positivamente como negativamente, o desempenho dos sistemas que venham a utilizar o *middleware* PerSec para implementar segurança fim a fim.

a aplicação do *middleware* PerSec para prover a segurança fim a fim também reduz bastante o número de elementos XML relacionados à segurança na estruturação das mensagens SOAP, se comparado à aplicação das especificações XML *Signature* e XML *Encryption* para a mesma finalidade. Isso implica um melhor desempenho no processamento dessas mensagens, já que reduz a complexidade de sua estrutura, porque a inserção de novos elementos XML a uma mensagem SOAP aumenta sua complexidade e, consequentemente, seu custo computacional, como apontam os trabalhos de Chen et al. (2007), Liu et al. (2005) e Andresen et al. (2004), já discutidos na revisão bibliográfica realizada neste estudo.

6.1 TRABALHOS FUTUROS

A perspectiva para trabalhos futuros, em relação ao trabalho efetuado neste estudo, consiste em:

a) expandir os níveis de confidencialidade do *middleware* PerSec para trabalhar com ouros algoritmos criptográficos, a fim de testá-los em termos de desempenho e segurança;
b) desenvolver uma nova versão do PerSec que seja capaz de interagir com *Web Services* desenvolvidos em qualquer linguagem de programação, visando tornar o *middleware* uma ferramenta universal para esse tipo de aplicação;
c) realizar testes com diferentes configurações em um ambiente de rede distribuído com diversas aplicações clientes e tamanhos distintos de mensagens;
d) automatizar a definição dos níveis de confidencialidade do *middleware* PerSec. Isto poderia ser feito por meio de uma camada de Inteligência Artificial ou, até mesmo, por meio de agentes baseados em *Web* Semântica, que seriam capazes de identificar dados

sensíveis quanto à segurança em uma mensagem SOAP, consequentemente, estabelecendo-os em níveis adequados de confidencialidade, a fim de serem criptografados pelo PerSec;

e) disponibilizar o *middleware* PerSec como uma aplicação *cloud computing* (computação nas nuvens ou computação em nuvem), para que seja possível utilizá-lo em qualquer lugar, independentemente de plataforma, por meio da *Internet*, sem a necessidade de instalação do sistema;
f) realizar um estudo de caso comparativo com soluções que empregam as especificações de segurança XML *Encryption* e *Signature*, como a *WS-Security*.

6.2 PUBLICAÇÕES

Deste trabalho se derivou um artigo, intitulado “PERSEC – *Middleware* for Multiple *Encryption* in *Web Services*” (ISBN: 978-1-4799-8827-3). Esse artigo foi aceito na Conferência Internacional: *Information Technology – New Generations (ITNG)*, 2015, e está disponível na biblioteca digital *IEEE Xplore*, no endereço: http://ieeexplore.ieee.org/.

REFERÊNCIAS

ALBUQUERQUE, R.; RIBEIRO, B. **Segurança no Desenvolvimento de *Software*** – Como desenvolver sistemas seguros e avaliar a segurança de aplicações desenvolvidas com base na ISO 15.408. Rio de Janeiro: Campus, 2002.

ALONSO, G. et al. **Web Services**: Concepts, Architectures and Applications. Springer, Berlin, 1. ed., Outubro, 2003. 354p.

ALROUH, B.; GHINEA, G. A performance evaluation of security mechanisms for *Web Services*. In: INTERNATIONAL CONFERENCE ON INFORMATION ASSURANCE AND SECURITY, 5., 2009, Washington. **Proceedings…**Washington, DC, USA: IEEE Computer Society, 2009, p. 715-718.

ANDRESEN, D. et al. Lye: A high-performance caching soap implementation. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, ICPP, 1., 2004, **Proceedings…** 2004. p. 143-150.

APACHE. **Apache tomcat**. The Apache Software Foundation. Disponível em: <http://tomcat.apache.org/>. Acesso em: 25 jun. 2015.

BURNETT, S.; PAINE, S. **Criptografia e segurança:** o guia oficial RSA. Rio de Janeiro: Campus, 2002.

CANYANG, K.; BOOTH, D. **Web Services Description Language (WSDL) version 2.0**. Technical Report, W3C. [S.l.]: [s.n.], 2007.

CHEN, S. et al. Performance evaluation and modeling of Web Services security. In: IEEE INTERNATIONAL CONFERENCE ON WEB SERVICES, 2007. **Proceedings…** 2007. p. 431-438.

DAEMEN, J.; RIJMEN, V. The block cipher rijndael. In: CARDIS'98: OF THE INTERNATIONAL CONFERENCE ON SMART CARD RESEARCH AND APPLICATIONS, 2000, London, UK. **Proceedings…** 2000. p. 277-284.

DIFFIE, W.; HELLMAN, M. E. **New directions in cryptography**. In: IEEE TRANSACTIONS ON INFORMATION THEORY, v. IT-22, n. 6, 1976, p. 644-654.

ENGELEN, R.; ZHANG, W. Identifying opportunities for Web Services security performance optimizations. In: IEEE CONGRESS ON SERVICES. Part I, 2008. **Proceedings…** 2008a. p. 209-210.

_____. An overview and evaluation of Web Services security performance optimizations. In: IEEE INTERNATIONAL CONFERENCE ON WEB SERVICES, 2008. ICWS'08. **Proceedings…** 2008b. p. 137-144.

ERL, T. **Service-oriented architecture:** concepts, technology, and design. [S.l.]: Prentice Hall PTR, 2005. p.792.

FELDHOFER, M.; DOMINIKUS, S.; WOLKERSTORFER, J. Strong authentication for rfid systems using the aes algorithm. In: WORKSHOP ON CRYPTOGRAPHIC HARDWARE AND EMBEDDED SYSTEMS. CHES, 2004, **Proceedings…** 2004. p. 357-370.

GOMES, D. A. **Web Services SOAP em Java**: guia prático para o desenvolvimento de *Web Services* em Java. São Paulo: Novatec, 2009.

GRUSCHKA, N. et al. Server-side streaming processing of ws-security. **IEEE Transactions On Services Computing**, n. 99, p. 1-14, 2011.

GUDGIN, M. et al. **SOAP version 1.2. Part 1**.**Technical Report**. Vancouver: W3C, 2007.

IBM. **JAX-WS**. Disponível em: <http://www-01.ibm.com/ support/ knowledgecenter/ SS4JCV_7.5.5/ com.ibm.webservice.wsfp.doc/ topics/ cjaxws.html?lang=en >. Acesso em: 27 maio 2015.

JAIN, R. K. **The art of computer systems performance analysis**: techniques for experimental design, measurement, simulation, and modeling. N. York: Wiley, 1991.

JDOM.ORG. **JDOM**. Disponível em: <http://www.jdom.org>. Acesso em: 27 maio 2015.

JENSEN, M.; GRUSCHKA, N.; HERKENHONER, R.; LUTTENBERGER, N. **SOA and Web Services:** New technologies, new standards - new attacks. In: ECOWS ’07: EUROPEAN CONFERENCE ON WEB SERVICES, 5., 2007, Washington, DC, USA. **Proceedings…** Washington, DC, USA: IEEE Computer Society, 2007, p. 35-44.

JOSUTTIS, N. **Soa in practice**: The art of distributed system design. Sebastopol: O’Reilly Media, Inc., 2007.

JURIC, M. B. et al. Comparison of performance of Web Services, ws-security, rmi, and rmi-ssl. **The Journal of Systems and Software**, v. 79, 2006, p. 689-700.

KANNEGANTI, R.; CHODAVARAPU, P. **Soa security**. Greenwich, CT, USA: Manning Publications Co., 2008.

KNAP, T.; MLÝNKOVÁ, I. Towards more secure Web Services: Pitfalls of various approaches to xml Signature verification process. In: ICWS ’09: 2009 IEEE INTERNATIONAL CONFERENCE ON WEB SERVICES, 2009, Washington, DC, USA. **Proceedings…** Washington, DC, USA: IEEE Computer Society, 2009. p. 543-550.

KOCHER, P.; e al. Security as a new dimension in embedded system design. In: DAC ’04: ANNUAL DESIGN AUTOMATION CONFERENCE, 41., 2004, New York - NY. **Proceedings…** New York - NY, USA: ACM, 2004. p. 753-760.

KRAUSE, M.; TIPTON, H. F. **Handbook of Information Security Management**. N. York: Auerbach Publications, 1999.

KUROSE, J. F.; ROSS, K. W. **Computer networking**: A top-down approach. 5. ed. Boston: Addison Wesley, 2009.

LIU, H.; PALLICKARA, S.; FOX, G. Performance of web service security. In: ANNUAL MARDI GRAS CONFERENCE, 13., 2005, Baton Rouge, Louisiana. **Proceedings…** 2005. p. 1-8.

MASHOOD, M.; WIKRAMANAYAKE, G. Architecting secure web services through policies. In: INTERNATIONAL CONFERENCE ON INDUSTRIAL AND INFORMATION SYSTEMS, ICIIS 2007. 2007, Sri Lanka. **Proceedings…** 2007. p.5-10.

MICROSOFT. **Security in a Web Services world**: A proposed architecture and roadmap. 2002. Disponível em: <http://msdn.microsoft.com/men-us/library/ ms977312.aspx >. Acesso em: 25 maio 2015.

Windows 8.1, Instructions. 2015. Disponível em: <http://windows.microsoft.com/pt-br/windows-8/whats-new >. Acesso em: jun. 2015.

TECHNET LIBRARY. **How RPC Works**. 2015. Disponível em: <https://technet.microsoft.com/pt-br/library/Cc738291(v=WS.10).aspx> Acesso em: 17 jun. 2015.

MOGOLLON, M. **Cryptography and security services**: mechanisms and applications. Dallas: IGI Global, 2008.

MOHAMMED, A. R.; ANDREAS S. SOAP-based Secure Conversation and Collaboration. In: INTERNATIONAL CONFERENCE ON WEB SERVICES (ICWS), 2007, Sri Lanka. **Proceedings…** 2007.

MORENO, E. D.; PEREIRA, F. D.; CHIARAMONTE, R. B. **Criptografia em software e hardware**. São Paulo: Novatec, 2005.

NAVYA, S.; JIGANG, L. A Framework for Enhancing Web Services Security. In: ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE (COMPSAC), 31., Beijing, 2007. **Proceedings…** 2007.

NAGAPPAN, R.; SKOCZYLAS, R.; SRIGANESH, R. P. **Developing java Web Services**. New York, USA: John Wiley & Sons, Inc., 2003.

NORDBOTTEN, N. A. Xml and Web Services security standards. **IEEE Communications Surveys Tutorials**, v. 11, n. 3, p. 4-21, 2009.

OLIVEIRA, E. J. S. **Comunicação segura e confiável para sistemas multiagentes adaptando especificações XML**. 2006. Disponível em: <http://www.dominiopublico.gov.br/pesquisa/DetalheObraForm.do?select_action=&co_obra=34975 >. Acesso em: 20 jun. 2014.

O'NEILL, M. **Web services security**. New York, USA: McGraw-Hill, Inc., 2003.

ORACLE. **Java SE Security.** Ano. Disponível em: <http://_www.oracle.com/ technetwork/ java/ javase/ tech/ index-jsp-136007.html.>. Acesso em: 27 maio 2015.

POSTGREESQL. **E.5. Release 9.4**. 2014. Disponível em: <http://www.postgresql.org/docs/9.4/static/release-9-4.html >. Acesso em: 17 jun. 2015.

RODRIGUES, D.; BRANCO, K. R. L. J. C. **Avaliação de desempenho de *web services* seguros:** um estudo comparativo. 2009. Disponível em: <http:// lasdpc.icmc.usp.br/ disciplinas/ pos-graduacao/ avaliacao-de-desempenho- / 2009/ trabalho-final/ aval% 20douglas.pdf/view>. Acesso em: 17 jun. 2014.

ROSENBERG, J.; REMY, D. **Securing Web Services with ws-security**: Demystifying ws-security, ws-policy, saml, xml *Signature*, and xml *Encryption*. U.K.: Pearson Higher Education, 2004.

SAMPIERI, R. H.; COLLADO, C. F.; LÚCIO, P. B. **Metodologia de pesquisa**. 3. ed. São Paulo: McGraw-Hill Interamericana do Brasil Ltda., 2006.

SIDDIQUI, B. **Exploring XML Encryption, part 1**. IBM Corporation. 2002. Disponível em: <http://www.ibm.com/developerworks/xml/library/x-encrypt/>. Acesso em: nov. 2014.

SILVA, R. F., CUNHA, J. A. **Arquitetura de segurança em aplicações baseadas em *web services***. 2005. Disponível em: <http:// www2.ifrn.edu.br/ ojs/index.php/ HOLOS/ article/ view/ 77/ 82>. Acesso em: 18 jun. 2015.

SNELL, J.; TIDWELL,D. ; KULCHENKO, P. **Programming Web Services with SOAP**. 1. ed. [S.l.]: O'Reilly Media, Inc., dec. 2001. 264p.

SOSNOSKI, D. **Java *Web Services***: Axis2 ws-security signing and *Encryption*. IBM Corporation. 2009. Disponível em: <http:// www.ibm.com/ developerworks/ java/ library/ j-jws5/>. Acesso em: 12 jul. 2015.

SOUZA, S.C. de. **Segurança para *Web Services* com criptografia heterogênea baseada em Proxy**. 2010. Dissertação (Mestrado) – Pontifícia Universidade Católica do Rio Grande do Sul. Porto Alegre, 2010.

TANENBAUM, A. S. **Computer networks**. N. J., USA: Prentice Hall Professional Technical Reference, 2003.

WEIS, B. **The use of rsa/sha-1 Signatures within encapsulating security payload (esp) and authentication header (ah)**. RFC 4359. 2006. Disponível em: <http://www.ietf.org/rfc/rfc4359.txt>. Acesso em: 9 abr. 2014.

W3C. **Web Services architecture**. 2004. Disponível em: <http://www.w3.org/TR/ws-arch/>. Acesso em: 25 jan. 2014.

XML *Encryption* syntax and processing. 2002. Disponível em: <http://www.w3.org/TR/xmlenc-core/>. Acesso em: 15 maio 2014.

XML Signature syntax and processing (second edition). 2008b. Disponível em: <http://www. w3.org/TR/xmldsig-core/>. Acesso em: 15 maio 2014.

YAMANY, H.F. EL; CAPRETZ, M. A. M.; ALLISON, D. S. **Quality of Security Service for Web Services within SOA**. 2009. Disponível em: <http://www.computer.org/csdl/proceedings/services/2009/3708/00/3708a653-abs.html>. Acesso em: 15 dez. 2014.

YAMANY, H.F. EL; CAPRETZ, M. A. M. **Use of Data Mining to Enhance Security for SOA**. 2008. Disponível em: <http:// www.computer.org/ csdl/ proceedings/iccit/ 2008/ 3407/ 01/ 3407a551- abs.html >. Acesso em: 15 dez. 2014.

YAU, S. S.; YIN, Y.; AN, H. G. An adaptive tradeoff model for service performance and security in service-based systems. In: ICWS '09: PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON WEB SERVICES, 2009, Washington, DC, USA. **Proceedings…** Washington, DC, USA: IEEE Computer Society, 2009.

YIN-SOON, L. et al. Design and Implementation of an XMLFirewall. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE AND SECURITY, 2006. Guangzhou, China. **Proceedings…** 2006.

YING, Y.; HUANG, Y.; WALKER, D. W. A performance evaluation of using soap with attachments for e-science. In: UK E-SCIENCE ALL HANDS MEETING. Nottingham, UK, 2005, p.796-803.

YUE-SHENG, G.; BAO-JIAN, Z.; WU, X. Research and realization of *Web Services* security based on xml *Signature*. In: OF INTERNATIONAL CONFERENCE ON NETWORKING AND DIGITAL SOCIETY, 2., 2009. **Proceedings…** Guiyang, Guizhou, China, 2009, p. 116-118.

ZHANG, D.; CODDINGTON, P.; WENDELBORN, A. Binary data transfer performance over highlatency networks using web service attachments. In: INTERNATIONAL CONFERENCE ON E-SCIENCE AND GRID COMPUTING, 2007, Bangalore, Índia. **Proceedings…** 2007. p. 261-269.

APÊNDICE A – DOCUMENTAÇÃO JAVADOC DO *MIDDLEWARE* PERSEC

Title bold -this the title

Package persec

Class Summary

Cipher

MessageUtils

Signature

Summarizer

---

persec

Class Cipher

```
java.lang.Object
    |
    +--persec.Cipher
```

< Constructors > < Methods >

---

public class **Cipher**
extends java.lang.Object

Constructors

Cipher

```
public  Cipher()
```

Methods

clearKeys

```
public void clearKeys()
```

---

decrypt

```
public java.lang.String decrypt(java.lang.String content)
```

---

decrypt

```
public java.lang.String decrypt(java.lang.String content,
                                int level)
```

---

decrypt

```
public void decrypt(javax.xml.soap.SOAPMessage message)
```

---

encrypt

```
public java.lang.String encrypt(java.lang.String content,
                                int level)
```

---

encrypt

```
public void encrypt(javax.xml.soap.SOAPMessage message,
                    java.lang.String[] tags,
                    int level)
```

---

generateKey

```
public boolean generateKey(int level)
```

---

getKey

```
public java.lang.String getKey(int level)
```

Title bold -this the title

main

```
public static void main(java.lang.String[] args)
```

---

setKey

```
public void setKey(java.lang.String encodedKey,
                   int level)
```

---

persec

Class MessageUtils

```
java.lang.Object
    |
    +--persec.MessageUtils
```

< Constructors > < Methods >

public class **MessageUtils**
extends java.lang.Object

Constructors

MessageUtils

```
public  MessageUtils()
```

Methods

getMessageAsByteArray

```
public static byte[] getMessageAsByteArray(javax.xml.soap.SOAPMessage message)
```

---

printMessage

```
public static void printMessage(javax.xml.soap.SOAPMessage message)
```

Title bold -this the title

persec

Class Signature

```
java.lang.Object
    |
    +--persec.Signature
```

< Constructors > < Methods >

public class **Signature**
extends java.lang.Object

Constructors

Signature

```
public  Signature()
```

Methods

createSignature

```
public static byte[] createSignature(java.security.PrivateKey key,
                                     byte[] buffer)
```

getPrivateKeyFromFile

```
public static java.security.PrivateKey
getPrivateKeyFromFile(java.io.InputStream cert,
                                                    java.lang.String
alias,
                                                    java.lang.String
password)
```

Title bold -this the title

getPublicKeyFromFile

```
public static java.security.PublicKey getPublicKeyFromFile(java.io.InputStream
cert,
                                                           java.lang.String
alias,
                                                           java.lang.String
password)
```

---

getPublicKeyFromFileX509

```
public static java.security.PublicKey
getPublicKeyFromFileX509(java.io.InputStream cert)
```

---

sign

```
public static void sign(javax.xml.soap.SOAPMessage message,
                        java.security.PrivateKey key)
```

---

verifySignature

```
public static boolean verifySignature(java.security.PublicKey key,
                                      byte[] buffer,
                                      byte[] signed)
```

---

verifySignature

```
public static boolean verifySignature(javax.xml.soap.SOAPMessage message,
                                      java.security.PublicKey key)
```

---

persec

Class Summarizer

```
java.lang.Object
    |
    +--persec.Summarizer
```

< Constructors > < Methods >

Title bold -this the title

public class **Summarizer**
extends java.lang.Object

Constructors

Summarizer

```
public  Summarizer()
```

Methods

summarize

```
public static void summarize(javax.xml.soap.SOAPMessage message)
```

---

verifySummary

```
public static boolean verifySummary(javax.xml.soap.SOAPMessage message)
```

APÊNDICE B – DOCUMENTAÇÃO JAVADOC DA APLICAÇÃO *BANKINGAPP*

Title bold -this the title

Package banking.client

Class Summary

AccountClient

ClientHandler

---

banking.client

Class AccountClient

```
java.lang.Object
    |
    +--banking.client.AccountClient
```

< Constructors > < Methods >

public class **AccountClient**
extends java.lang.Object

Constructors

AccountClient

```
public  AccountClient()
```

Methods

main

```
public static void main(java.lang.String[] args)
```

Title bold -this the title

banking.client

Class ClientHandler

```
java.lang.Object
    |
    +--banking.client.ClientHandler
```

All Implemented Interfaces:
javax.xml.ws.handler.soap.SOAPHandler

< Constructors > < Methods >

public class **ClientHandler**
extends java.lang.Object
implements javax.xml.ws.handler.soap.SOAPHandler

Constructors

ClientHandler

```
public  ClientHandler()
```

Methods

close

```
public void close(javax.xml.ws.handler.MessageContext context)
```

getHeaders

```
public java.util.Set getHeaders()
```

handleFault

```
public boolean handleFault(javax.xml.ws.handler.soap.SOAPMessageContext
context)
```

Title bold -this the title

handleMessage

```
public boolean handleMessage(javax.xml.ws.handler.soap.SOAPMessageContext
context)
```

Title bold -this the title

Package banking.services

Interface Summary

AccountServicesInterface

Class Summary

AccountServices

Balance

---

banking.services

Class AccountServices

```
java.lang.Object
    |
    +--banking.services.AccountServices
```

All Implemented Interfaces:
AccountServicesInterface

< Constructors > < Methods >

---

public class **AccountServices**
extends java.lang.Object
implements AccountServicesInterface

Constructors

AccountServices

```
public  AccountServices()
```

Methods

getBalance

```
public Balance getBalance(java.lang.String agency,
                          java.lang.String number,
                          java.lang.String password)
```

Title bold -this the title

banking.services

Interface AccountServicesInterface

< Methods >

public interface **AccountServicesInterface**

Methods

getBalance

```
public Balance getBalance(java.lang.String agency,
                          java.lang.String number,
                          java.lang.String password)
```

banking.services

Class Balance

```
java.lang.Object
    |
    +--banking.services.Balance
```

< Constructors > < Methods >

public class **Balance**
extends java.lang.Object

Constructors

Balance

```
public  Balance()
```

Title bold -this the title

Balance

```
public  Balance(java.lang.String name,
                java.util.Date date,
                float balance)
```

Methods

getBalance

```
public float getBalance()
```

---

getDate

```
public java.util.Date getDate()
```

---

getName

```
public java.lang.String getName()
```

---

setBalance

```
public void setBalance(float balance)
```

---

setDate

```
public void setDate(java.util.Date date)
```

---

setName

```
public void setName(java.lang.String name)
```

Title bold -this the title

toString

```
public java.lang.String toString()
```

Overrides:

toString in class java.lang.Object

Title bold -this the title

Package banking.services.handler

Class Summary

ServerHandler

banking.services.handler

Class ServerHandler

```
java.lang.Object
    |
    +--banking.services.handler.ServerHandler
```

All Implemented Interfaces:
javax.xml.ws.handler.soap.SOAPHandler

< Constructors > < Methods >

public class **ServerHandler**
extends java.lang.Object
implements javax.xml.ws.handler.soap.SOAPHandler

Constructors

ServerHandler

```
public  ServerHandler()
```

Methods

close

```
public void close(javax.xml.ws.handler.MessageContext context)
```

getHeaders

```
public java.util.Set getHeaders()
```

handleFault

```
public boolean handleFault(javax.xml.ws.handler.soap.SOAPMessageContext
context)
```

---

handleMessage

```
public boolean handleMessage(javax.xml.ws.handler.soap.SOAPMessageContext
context)
```

Title bold -this the title

Package banking.services.jaxws

Class Summary

GetBalance

GetBalanceResponse

---

banking.services.jaxws

Class GetBalance

```
java.lang.Object
    |
    +--banking.services.jaxws.GetBalance
```

< Constructors > < Methods >

---

public class **GetBalance**
extends java.lang.Object

Constructors

GetBalance

```
public  GetBalance()
```

Methods

getAgency

```
public java.lang.String getAgency()
```

Returns:

returns String

Title bold -this the title

getNumber

```
public java.lang.String getNumber()
```

Returns:

returns String

---

getPassword

```
public java.lang.String getPassword()
```

Returns:

returns String

---

setAgency

```
public void setAgency(java.lang.String agency)
```

Parameters:

agency - the value for the agency property

---

setNumber

```
public void setNumber(java.lang.String number)
```

Parameters:

number - the value for the number property

---

setPassword

```
public void setPassword(java.lang.String password)
```

Parameters:

password - the value for the password property

Title bold -this the title

banking.services.jaxws

Class GetBalanceResponse

```
java.lang.Object
    |
    +--banking.services.jaxws.GetBalanceResponse
```

< Constructors > < Methods >

public class **GetBalanceResponse**
extends java.lang.Object

Constructors

GetBalanceResponse

```
public  GetBalanceResponse()
```

Methods

getReturn

```
public Balance getReturn()
```

Returns:

returns Balance

setReturn

```
public void setReturn(Balance _return)
```

Parameters:

_return - the value for the _return property