Universidade Federal do Maranhão
Centro de Ciências Exatas e Tecnologia
Curso de Pós-graduação em Engenharia de Eletricidade

Emerson José Santos Oliveira

Comunicação Segura e Confiável para Sistemas Multiagentes Adaptando Especificações XML

São Luís
2006

Emerson José Santos Oliveira

Comunicação Segura e Confiável para Sistemas Multiagentes Adaptando Especificações XML

Dissertação apresentada ao Curso de Pós-graduação em Engenharia de Eletricidade da UFMA, como requisito para a obtenção do grau de MESTRE em Engenharia de Eletricidade, Área de Concentração: Ciência da Computação.

Orientador: Zair Abdelouahab

Ph.D. em Ciência da Computação - UFMA

São Luís

2006

Oliveira, Emerson José Santos

Comunicação Segura e Confiável para Sistemas Multiagentes Adaptando Especificações XML / Emerson José Santos Oliveira - São Luis, 2006

114f.

Dissertação (Mestrado em Engenharia de Eletricidade) - Universidade Federal do Maranhão, 2006.

1.Sistemas Multiagentes 2.Comunicação Segura 3.Comunicação Confiável 4.XKMS 5.XML-Enc 6.XML-DSig 7.WS-RM. I.Título.

CDU 004.891

COMUNICAÇÃO SEGURA E CONFIÁVEL PARA SISTEMAS MULTIAGENTES ADAPTANDO ESPECIFICAÇÕES XML

Emerson José Santos Oliveira

Dissertação aprovada em 01 de dezembro de 2006.

Prof. Zair Abdelouahab, Ph. D.
(Orientador)

Prof. Dalton Dario Serey Guerrero, Dr.
(Membro da Banca Examinadora)

Prof. Sofiane Labidi, Dr.
(Membro da Banca Examinadora)

Denivaldo Cícero Pavão Lopes, Dr.
(Membro da Banca Examinadora)

À minha mãe, pelo apoio e amor incondicional, não só no período deste trabalho, mas em toda minha vida.

Agradecimentos

A Deus, pelo apoio nos muitos momentos de desânimo, pelo atendimento de minhas preces e pela iluminação em todas as minhas escolhas durante este período.

A minha mãe, por tudo que tem feito no período deste trabalho e em toda minha vida.

Ao meu orientador, Prof. Ph.D. Zair Abdelouahab, pela paciência e orientação na realização deste trabalho.

Ao Pesquisador Dr. Denivaldo Lopes, pelo apoio durante a dissertação.

A todos os professores do Curso de Pós-Graduação em Engenharia de Eletricidade que não mediram esforços para transmitir aos seus alunos o conhecimento necessário.

A FAPEMA pelo financiamento da publicação de um artigo no DEXA-NBiS 2006.

A Wagner pelo apoio técnico no gerenciamento e manutenção da infraestrutura computacional usada nesta pesquisa.

Aos meus superiores dos empregos que tive durante o período deste trabalho que me concederam licenças para dedicar-me ao mestrado.

A Emanoel Claudino pela companhia nos estudos e no desenvolvimento dos projetos finais das disciplinas e à sua esposa, Arabela, pela paciência e apoio.

A todos aqueles que freqüentam o Laboratório de Sistemas Distribuídos do Prof. Zair pela companhia em vários sábados de trabalho.

Agradeço a todos que, de uma forma ou de outra, contribuíram para a realização deste trabalho e me ajudaram no decorrer do curso de pós-graduação.

"Ninguém se engane a si mesmo. Se alguém dentre vós se julga sábio à maneira deste mundo, faça-se louco para tornar-se sábio, porque a sabedoria deste mundo é loucura diante de Deus; pois (diz a Escritura) Ele apanhará os sábios na sua própria astúcia."

(ICor 3:18,19)

Resumo

Sistemas multiagentes estão evoluindo em aplicações corporativas e estão sendo utilizados cada vez mais em ambientes abertos, como a Internet. Vários tópicos devem ser observados nesta evolução, como a segurança e a confiança na entrega das mensagens. Neste trabalho, nós propomos um modelo de comunicação segura e um modelo de entrega de mensagens confiável. Os dois modelos têm tecnologias XML adaptadas, que consistem em várias especificações XML (*Extensible Markup Language*) e no padrão RDF (*Resource Description Framework*). Para a segurança, várias especificações foram adaptadas: 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*). Para a parte de confiabilidade de comunicação, a especificação WS-RM (*WS-ReliableMessaging*) foi adaptada para garantir a entrega das mensagens. O padrão RDF foi utilizado para habilitar os agentes a trocarem mensagens usando a sintaxe XML. Neste trabalho, os testes com os protótipos das soluções propostas e as comparações com algumas soluções existentes também são apresentados.

Palavras-chave: Sistemas multiagentes, Segurança, Comunicação Confiável, Especificações XML, RDF.

Abstract

Multi-agent systems are evolving to enterprise applications and they are more used in open environments, such as the Internet. Many issues should be considered with this evolution, like security and reliability of communication. In this work, we propose a solution for secure communication and a solution for a reliable communication; both solutions are for multi-agent systems. These solutions adapt XML technologies and consist of several XML Specifications and RDF standard. For secure communication, we adapt the XML-DSig specification to provide integrity and digital signature; the XML-Enc specification is used to provide confidentiality through encryption, and the XKMS specification is used to provide PKI support. For reliable communication, we adapt the WS-RM specification that guarantees the message delivery. The RDF standard enables agents for exchanging messages using XML syntax. In this work, the tests with prototypes of the proposed solutions and comparisons with other solutions are also presented.

Keywords: Multi-Agent Systems, Security, Reliable Communication, XML Security Specifications, RDF.

Lista de Figuras

Lista de Siglas

IP	Internet Protocol
MAC	Media Access Control
3DES	Triple Data Encryption Standard
ACL	Agent Communication Language
AID	Agente Identifier
AMS	Agent Management System
API	Application Programming Interface
ARP	Address Resolution Protocol
CA	Certification Authority
CL	Content Language
DES	Data Encryption Standard
DF	Directory Facilitator
FIPA	Foundation for Intelligent Physical Agents
HTTP	HyperText Transfer Protocol
IA	Inteligência Artificial
IBM	International Business Machines Corporation
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
J2EE	Java 2 Platform, Enterprise Edition
JADE	Java Agent DEvelopment Framework
JAXB	Java Architecture for XML Binding
MAS	Multi-Agent System
MITM	Man-in-the-Middle
MTP	Message Transport Protocol
NIDIA	Network Intrusion Detection System based on Intelligent Agents
OASIS	Organization for the Advancement of Structured Information Standards
OO	Orientado a Objetos
PKI	Public Key Infrastructure
POO	Projeto Orientado a Objetos

RDF	Resource Description Framework
RPC	Remote Procedure Call
RSA	Rivest-Shamir-Adleman
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TCP/IP	Transport Control Protocol/Internet Protocol
TI	Tecnologia da Informação
TSIK	Trust Service Integration Kit
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF-8	Unicode Transformation Format 8-bit
W3C	World Wide Web Consortium
WS-RM	WS-ReliableMessaging
WSDL	Web Services Description Language
X-DSig	XML Digital Signature
X-Enc	XML Encryption
XKMS	XML Key Management Specification
XML	eXtensible Markup Language
XSLT	eXtesible Stylesheet Language Transformations

Lista de Listagens

Sumário

1 Introdução

Um sistema multiagente ou MAS (Sigla em inglês para *Multi-Agent System*) corresponde a um tipo de sistema distribuído que consiste de várias entidades autônomas chamadas agentes. Os agentes são softwares que têm autonomia suficiente e "inteligência" para realizarem várias tarefas com pouca ou nenhuma intervenção humana. Os agentes são bastante independentes, mas também podem interagir com outros agentes, sistemas externos e humanos para completar determinadas tarefas [43].

Anteriormente, os sistemas multiagentes consistiam em sistemas proprietários e fechados, entretanto, atualmente, estes sistemas têm sido utilizados largamente em redes abertas [29]. Quando utilizados em redes abertas, como a Internet, os agentes podem acessar serviços remotos, localizar informações na Internet e realizar transações comerciais. Sistemas corporativos como comércio eletrônico para a Internet têm sido a maior área de aplicação dos MASs [29]. Infelizmente, a Internet não pode ser considerada um ambiente seguro.

Entidades externas possuem várias maneiras de interagir de forma negativa (práticas de ataques) com um agente. Por exemplo, intrusos podem interceptar mensagens trocadas entre dois agentes para capturar informações sigilosas [8], [43]. Assim MASs, para a sua introdução efetiva em soluções corporativas, devem fornecer mecanismos de segurança, para que seus agentes possam se defender contra ataques de outros agentes e humanos. Assim, mecanismos como confidenciabilidade (sigilo), integridade, não-repúdio[1] e autenticação são indispensáveis para os sistemas multiagentes [8], [25], [43].

Outro pré-requisito para a introdução efetiva dos MASs em soluções corporativas é a capacidade fundamental de garantia de entrega das mensagens entre os vários participantes do sistema [3].

Soluções corporativas também devem ser robustas o suficiente para suportar garantias de recebimento de mensagens do sistema. Esta particularidade é muito importante para questões que envolvem transações comerciais.

Sistemas multiagentes devem fornecer então mecanismos de garantia de entrega

[1]Não-repúdio é a impossibilidade do remetente negar a autoria de uma mensagem.

de mensagens, assegurando a seus usuários robustez em questão de transações e de funcionalidades distribuídas críticas.

1.1 Cenário e Definição do Problema

Várias especificações de segurança para *Web Services* foram publicadas por várias organizações confiáveis, como a IETF [18] e a W3C [40].

Dentre as especificações XML existentes, nós destacamos a especificação XKMS (*XML Key Management Specification*) [14] que define protocolos para as operações de registro e de distribuição de chaves públicas, ou seja, suporte ao esquema PKI (*Public Key Infrastructure*). A especificação XKMS habilita usuários de *Web Services* a utilizarem chaves públicas e privadas por meio de um servidor XKMS.

Também destacamos a especificação XML-Enc (*XML Encryption*) [9] e a XML-DSig (*XML Signature*) [10] responsáveis, respectivamente, por fornecer padrões de criptografia (confidenciabilidade) e de assinatura digital (integridade da mensagem e autenticação do remetente) aos *Web Services*. Estas duas especificações agregam os conceitos de confiabilidade e integridade às mensagens SOAP[2] transmitidas entre o provedor e o usuário do *Web Service* [15], [26] e podem utilizar as chaves gerenciadas pela especificação XKMS nos procedimentos de criptografia e de assinatura digital.

Já que estas especificações têm seu foco na mensagem SOAP [15], [26], que se trata de um documento XML, as mesmas especificações podem ser adaptadas para fornecer mecanismos de segurança em esquemas de comunicação não segura que usem também mensagens XML.

A FIPA (*Foundation for Intelligent Physical Agents*) [12] é uma organização sem fins lucrativos que promove tecnologias baseadas em agentes e a interoperabilidade de seus padrões com outras tecnologias. Os padrões FIPA representam uma coleção de padrões e especificações que possuem o propósito de promover a interoperabilidade entre agentes heterogêneos e os serviços que eles disponibilizam.

Dentre as especificações de comunicação de agentes publicadas pela FIPA, as especificações de Linguagem de Conteúdo ou CL (sigla em inglês para *Content Language*)

[2]SOAP (*Simple Object Access Protocol*) consiste no padrão de mensagens utilizado pelos *Web Services*. A codificação de mensagens SOAP utiliza o padrão XML.

são padrões para a representação do conhecimento transportado nas mensagens trocadas entre agentes. Uma destas CLs utiliza o padrão RDF [5], [13], [21]. O padrão RDF propõe a XML como sua sintaxe de codificação. Entretanto, nenhuma das especificações FIPA atuais fornece qualquer mecanismo de segurança [12], [29].

Como a RDF-CL habilita os agentes a trocarem mensagens em XML, nós podemos adaptar as especificações XML citadas (XML-Enc, XML-DSig e XKMS) para fornecer mecanismos de segurança em sistemas multiagentes cujos agentes "conversem" utilizando o padrão RDF.

Existe uma especificação XML chamada WS-RM (*WS-ReliableMessaging*) [2] que fornece um protocolo que habilita uma variedade de técnicas de entrega confiável de mensagens (confiabilidade de comunicação) para *Web Services*.

Em MASs que utilizem o RDF, a garantia de entrega de mensagens pode ser obtida por meio da adaptação do WS-RM e de suas técnicas de entrega confiável de mensagens.

Assim, nós propomos duas soluções que adaptam especificações XML para sistemas multiagentes que utilizem o padrão RDF: uma solução de segurança e outra de garantia de entrega de mensagens. A vantagem de termos duas soluções (uma de segurança e outra de confiabilidade de comunicação) é a possibilidade de utilizarmos apenas uma por vez ou as duas juntas de acordo com as dependências de cada ambiente. A solução de segurança adapta as especificações XML de segurança XML-Enc, XML-DSig e XKMS para o fornecimento dos mecanismos de segurança para agentes; e a solução de confiabilidade de comunicação adapta os protocolos da especificação WS-RM para a garantia de entrega de mensagens em comunicação entre agentes.

Para a implementação do protótipo, nós escolhemos o *framework* JADE [20], um ambiente de desenvolvimento de sistemas multiagentes que fornece um *framework* completo de comunicação, de ciclo de vida de agentes, de monitoramento de execução, entre outras funcionalidades para sistemas multiagentes.

O protótipo da solução de segurança é implementado como uma API[3] que pode ser utilizada por qualquer usuário que utilize o *framework* JADE.

Verificamos que agentes implementados em JADE e executando em diferentes

[3]Uma API consiste em uma biblioteca de componentes de terceiros que encapsulam várias rotinas visando facilitar o desenvolvimento de aplicações.

máquinas trocam mensagens utilizando um MTP (*Message Transport Protocol*) que consiste em uma implementação de protocolos de transporte de mensagens. Os mecanismos de transporte de mensagens de um MTP são relativamente transparentes ao usuário. Assim, o protótipo da solução de comunicação confiável é implementado como um MTP que também pode ser utilizado por qualquer usuário do *framework* JADE.

1.2 Objetivo Geral e Específicos

Tendo em vista tudo o que já foi dito, esta dissertação tem por objetivo geral propor uma solução de comunicação segura para MASs utilizando o esquema RDF e adaptando as especificações XKMS, XML-Enc e XML-DSig ao RDF para obter mecanismos de integridade, confidenciabilidade, não-repúdio e autenticação. O modelo de comunicação também tem suporte ao esquema PKI.

Este trabalho ainda tem por objetivo apresentar uma proposta de segurança específica para o servidor XKMS, responsável pelo gerenciamento das chaves públicas.

O servidor XKMS e os demais agentes possuem um mecanismo de autenticação proposto neste trabalho que utiliza criptografia de chaves comuns geradas por *Diffie-Hellman*[4].

Também é objetivo desta dissertação apresentar uma solução de entrega confiável de mensagens (garantia de entrega) para MASs utilizando o esquema RDF e adaptando a especificação WS-RM e suas técnicas de entrega confiável de mensagens.

As duas soluções utilizam componentes do JADE e podem ser utilizadas por qualquer usuário do mesmo *framework*. Os protótipos das soluções de segurança e de confiabilidade também serão apresentados.

De modo mais específico, esperamos:

- Apresentar o contexto de estudo empregado para a concepção das soluções e do mecanismo de segurança do servidor XKMS;
- Apresentar a solução de comunicação segura para MASs descrevendo a modelagem do protótipo utilizando a abordagem POO (Projeto Orientado a Objetos);

[4]O Algoritmo do *Diffie-Hellman* habilita duas máquinas a criarem uma mesma chave secreta.

- Apresentar a solução de comunicação confiável com garantia de entrega de mensagens para MASs descrevendo a modelagem do protótipo utilizando a abordagem POO;
- Apresentar as soluções existentes para o problema de segurança e de comunicação confiável e seus comparativos com as soluções propostas neste trabalho;
- Apresentar o IDS NIDIA e utilizá-lo como estudo de caso para as soluções propostas.

1.3 Organização do Trabalho

Esta dissertação está organizada em 7 capítulos.

No Capítulo 1, nós apresentamos a introdução desta dissertação, bem como, o cenário que motivou sua realização e os objetivos.

No Capítulo 2, nós apresentamos o contexto de estudo desta dissertação, fornecendo uma visão geral de MASs e do *framework* JADE. Neste capítulo, nós também abordamos os algoritmos e esquemas de segurança que são utilizados pela solução de segurança.

No Capítulo 3, nós apresentamos as especificações XML adaptadas na nossa solução de segurança e de confiabilidade e as soluções existentes de segurança e confiabilidade para o JADE.

No Capítulo 4, nós apresentamos a solução de segurança proposta, com suas características, o esquema de segurança específico do servidor XKMS e a modelagem POO do protótipo que implementa a solução de segurança. Ilustramos o trabalho com um exemplo de utilização e um comparativo com uma solução de segurança existente.

No Capítulo 5, nós apresentamos a solução de confiabilidade proposta, com as características do MTP implementado, agregamos um pequeno manual prático para a escrita de um MTP-JADE, discutimos a modelagem POO do protótipo da solução de confiabilidade de comunicação, e fornecemos um exemplo de utilização. Também apresentamos um comparativo com uma solução de confiabilidade existente.

No Capítulo 6, nós apresentamos o NIDIA e o utilizamos neste trabalho como estudo de caso para testar as soluções propostas.

Finalmente, no Capítulo 7, apresentamos as considerações finais, destacando as contribuições deste trabalho, suas limitações e as propostas de trabalhos futuros.

2 Contexto de Estudo

Este capítulo tem por objetivo apresentar uma visão geral sobre sistemas multiagentes e apresentar o *framework* JADE. Neste capítulo, todos os algoritmos e esquemas dos mecanismos de segurança que serão referenciados no restante da dissertação também são apresentados.

2.1 Agentes e Sistemas Multiagentes

Um agente pode ser uma pessoa, uma máquina ou uma porção de software. A definição básica de um agente seria: "aquele que age". Entretanto, para desenvolvimento de sistemas de TI, tal definição é muito geral. Agentes relacionados a TI precisam de atributos adicionais [16]. Alguns destes atributos que os agentes podem possuir em várias combinações incluem:

- Autônomos ou independentes: são capazes de atuar sem intervenção direta e externa, ou seja, eles possuem um grau de controle de seu estado interno e de suas ações baseado em sua própria experiência;
- Interativos: comunicam-se com o ambiente e outros agentes;
- Adaptáveis: capazes de responder para outros agentes e/ou seu ambiente em algum grau. Algumas formas de adaptação permitem aos agentes modificarem seus comportamentos baseados em experiências;
- Sociais: realizam interações que são percebidas por relações de "prazer social" ou de amizade, onde os agente são amigáveis ou sociáveis;
- Móveis: capazes de se transportarem de um ambiente para outro;
- Agem como um *proxy*: podem atuar entre alguma coisa ou alguém, isto é, atuam pelo interesse de alguém, representando-o, ou pelo benefício deste alguém ou alguma entidade;
- Pró-ativos: são orientados a objetivos e decididos a alcançá-los;

- Inteligentes: possuem conhecimento (por exemplo, crenças, objetivos, planos e suposições) e interagem com outros agentes usando linguagem de símbolos;
- Racionais: capazes de escolher uma ação baseada em seus objetivos e o conhecimento que uma ação em particular vai trazer para aproximá-los de seus objetivos;
- Imprevisíveis: capazes de agir de modo que não são totalmente previsíveis, mesmo que todas as condições iniciais sejam conhecidas. Eles são capazes de comportamento não-determinístico;
- Contínuos: podem ser vistos como um processo executando de forma contínua;
- Possuem caráter: possuem uma personalidade fiel;
- Transparentes e responsáveis (*accountable*): devem ser transparentes quando necessários, mas devem fornecer um *log* de suas atividades sob demanda;
- Coordenados: capazes de realizar alguma atividade em um ambiente compartilhado com outros agentes. Atividades são freqüentemente coordenadas via algum planejamento ou algum outro mecanismo de gerência de processo (*workflow*);
- Cooperativos: capazes de cooperar com outros agentes para conseguir um objetivo comum;
- Competitivos: capazes de cooperar com outros agentes exceto quando o sucesso de um agente implica na falha de outros (o contrário de cooperativos);
- Robustos: capazes de tratar com erros e dados incompletos de forma robusta;
- Confiáveis: agregam-se as Leis da Robótica[1] e são verdadeiros.

[1] As Três Leis da Robótica foram elaboradas pelo escritor Isaac Asimov em seu livro de ficção "Eu, Robô" (*I, Robot*, 1950) que dirigem o comportamento dos robôs. As Leis da Robótica são:

1ª lei: um robô não pode fazer mal a um ser humano e nem, por inação, permitir que algum mal lhe aconteça;

2ª lei: um robô deve obedecer às ordens dos seres humanos, exceto quando estas contrariarem a primeira lei;

3ª lei: um robô deve proteger a sua integridade física, desde que com isto não contrarie as duas primeiras leis.

Mais tarde foi introduzida uma "lei zero": um robô não pode fazer mal a humanidade e nem, por inação, permitir que ela sofra algum mal. Desse modo, o bem da humanidade é primordial ao dos indivíduos.

Para o meio industrial, uma definição única não existe ainda. A maioria concorda que agentes voltados a sistemas de TI não são úteis sem pelo menos as três primeiras das propriedades listadas. Outros acham que agentes de TI devem possuir todas as propriedades listadas em vários níveis. No mínimo, um agente de TI geralmente é visto como sendo uma entidade autônoma que interage com seu ambiente. Em outras palavras, ele deve ser capaz de perceber seu ambiente através de sensores e agir de acordo com seu ambiente com componentes de ação. Os agentes encontrados em sistemas de TI podem executar como um *software*, um hardware, um robô, ou uma combinação deles.

Resumidamente, um agente de *software* é definido como uma entidade de *software* autônoma que pode interagir com seu ambiente. Em outras palavras, eles são agentes que são implementados usando software. Isto significa que eles são autônomos e podem reagir com outras entidades, incluindo humanos, máquinas e outros agentes de *softwares* em vários ambientes e através de varias plataformas.

Agentes de *softwares* são padrões de projeto para softwares. Ferramentas, linguagens e ambientes são desenvolvidos especificamente para suportar o padrão baseado em agentes. Entretanto, o padrão de projeto de agentes pode também ser implementado usando ferramentas, linguagens e ambientes orientados a objetos - ou qualquer outra ferramenta, linguagem e ambiente que seja capaz de suportar entidades de *softwares* que sejam autônomas, interativas e adaptativas. Ferramentas baseadas em agentes são preferíveis porque os padrões de projeto de agentes são inerentes ao software - melhor do que programadas explicitamente. Em outras palavras, a tecnologia de objetos pode ser usada para habilitar a tecnologia baseada em agentes, mas a natureza de adaptação, interação e autonomia requerida pelos agentes não é atualmente suportada dentro da tecnologia OO. Enquanto estas propriedades possam ser (e estão sendo) adicionadas para a abordagem OO, os padrões de projetos para agentes e softwares baseados em agentes não são totalmente e diretamente suportados[2].

Podemos dizer então que os sistemas multiagentes são sistemas compostos de agentes coordenados através de seus relacionamentos de uns com os outros [16]. Por exemplo, em uma cozinha, a torradeira "sabe" quando a torrada está pronta, e a cafeteira "sabe" quando o café está pronto. Entretanto, não há nenhum ambiente de cooperação

Um robô não pode ter poder de escolha, excepto que seja para salvar vidas humanas e que com isto não contrarie as duas primeiras leis.

[2]Deste ponto em diante, o termo agente vai significar agente de software.

aqui, apenas os agentes isolados. Em um ambiente de multiagentes, a cozinha torna-se mais que uma coleção de processadores. As aplicações podem estar interconectadas de um jeito que a cafeteira e a torradeira podem assegurar que o café e a torrada estarão prontos ao mesmo tempo.

Assim, sistemas multiagentes possuem as seguintes justificativas:

- Um agente poderia ser construído e fazer tudo, mas agentes assim podem representar um gargalo para a velocidade e manutenção da aplicação. Dividir a funcionalidade através de vários agentes fornece modularidade, flexibilidade e extensibilidade;
- Conhecimento especializado não é freqüentemente disponível para um agente único. O conhecimento disperso entre várias fontes (agentes) pode ser integrado para uma visão mais completa quando necessária;
- Aplicações que requerem computação distribuída são melhor suportadas por sistemas multiagentes. Assim, agentes podem ser projetados como componentes autônomos de granulalidade fina que agem em paralelo. Processamento concorrente e resolução de problemas podem fornecer soluções para vários problemas que, atualmente, nós os abordamos de forma linear. Sistemas multiagentes, então, fornecem a última palavra em tecnologia de componentes distribuídos.

Um ambiente para suportar sistemas multiagentes deve:

- Fornecer uma infra-estrutura específica de comunicação e protocolos de integração;
- Ser tipicamente aberto e não ter um projetista centralizado ou função de controle *top-down*;
- Conter agentes que são autônomos, adaptativos e coordenados.

2.2 *Framework* JADE

O *framework* JADE (*Java Agent DEvelopment Framework*) [20] consiste em um *middleware* que facilita o desenvolvimento de sistemas multiagentes. O JADE é composto por:

- Um ambiente de execução onde os agentes podem "viver" e que deve estar ativo em uma determinada máquina antes de que um ou mais agentes possam ser executados;
- Uma biblioteca de classes que os programadores podem utilizar (diretamente ou as especializam) para desenvolver seus próprios agentes;
- Um conjunto de ferramentas gráficas que permite a administração e o monitoramento das atividades dos agentes em execução.

2.2.1 Ambiente de Execução do JADE

Cada instância de um ambiente de execução do JADE é chamado de um *container*, já que ele contém vários agentes. O conjunto de *containers* ativos é chamado de uma plataforma. Um *container* especial chamado de *Main Container* deve estar sempre ativo em uma plataforma e todas os outros *containers* devem se registrar junto a ele logo que são iniciados. Em outras palavras, o primeiro *container* de uma plataforma deve ser o *Main Container* e todos os demais devem ser *containers* "normais" e devem saber onde encontrar seu *Main Container* para se registrarem nele.

Se outro *Main Container* é iniciado, ele irá constituir uma plataforma diferente para a qual novos *containers* normais podem se registrar. A Figura 2.1 ilustra os conceitos de *containers* e plataformas apresentando um cenário de duas plataformas compostas por 3 e 1 *containers*. Os agentes são identificados por um nome único e podem se comunicar independentemente de sua localização. Um agente pode se comunicar com outro em um mesmo *container*, como os agentes A2 e A3; em diferentes *containers* da mesma plataforma, como A1 e A2; ou em diferente plataformas, como A4 e A5.

Além da habilidade de aceitar requisições de registros de outros *containers*, um *Main Container* difere dos demais *containers* por possuir dois agentes especiais, o agente AMS (*Agent Management System*) e o agente DF (*Directory Facilitator*) que são iniciados automaticamente sempre que um *Main Container* é criado (Figura 2.1).

O agente AMS fornece um serviço de nomeação para garantir que cada agente na plataforma possui um nome único e possui autoridade na plataforma para criar e finalizar um agente. O agente DF fornece um serviço de páginas amarelas para que cada agente possa encontrar outros agentes fornecendo os serviços que está necessitando para obter seus objetivos.

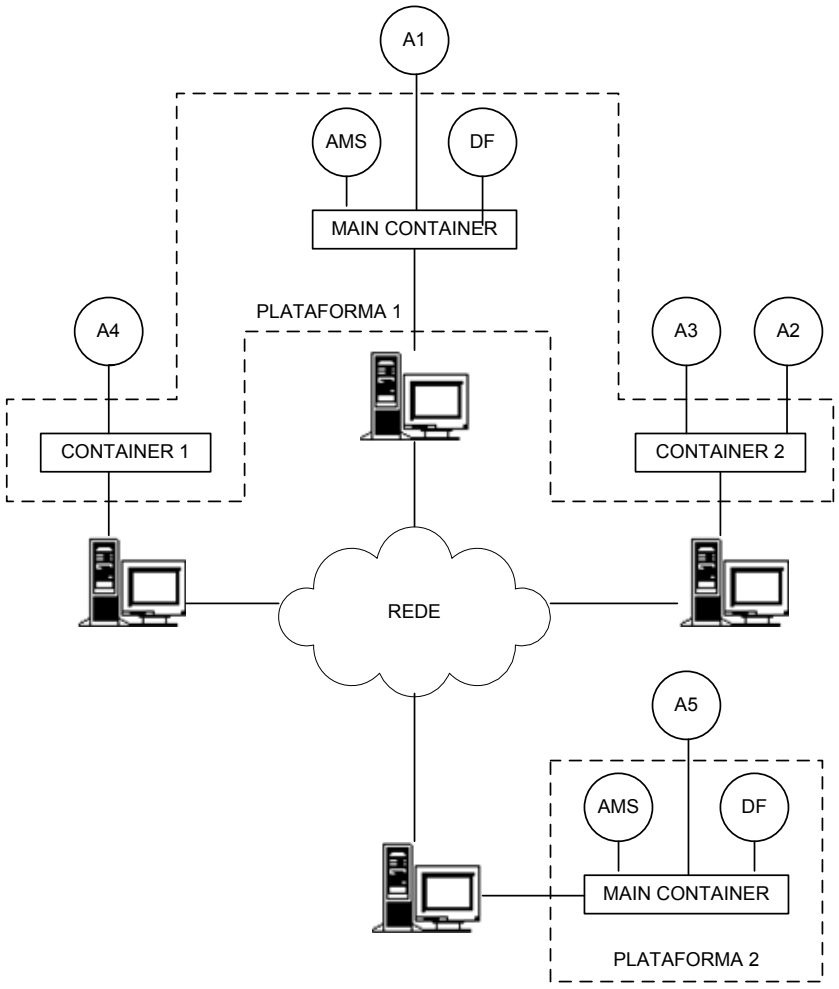


Figura 2.1: *Containers* e plataformas no JADE

2.2.2 Agentes em JADE

Um agente implementado em JADE deve estender a classe *jade.core.Agent*. A classe *Agent* fornece funcionalidades essenciais para que os agentes possam interagir com um plataforma, tais como métodos de registro, configuração e gerenciamento remoto do agente. A classe *Agent* possui o método *setup()*, onde é implementada a configuração inicial do agente, como a definição dos comportamentos do agente.

Os comportamentos de um agente são representados por instâncias de classes que implementam a classe *jade.core.behaviours.Behaviour*. A classe *Behavior* representa uma tarefa ou funcionalidade do agente. O método *action()* da classe *Behaviour* descreve as ações que são executadas por aquele comportamento do agente. A Figura 2.2 representa um agente e seu comportamento estendendo as classes *Agent* e *Behaviour*.

Cada agente executando em uma plataforma é identificada por um "identificador de agente" (ou AID de *Agent Identifier*) que é representado por uma

instância da classe *jade.core.AID*. O método *getAID()* da classe *Agent* permite obter o identificador do agente. Um objeto *AID* inclui um nome único global seguido de uma quantidade de endereços. O nome global de um agente tem o formato `<nome do agente>@<nome da máquina>:<porta>/JADE`, assim um agente chamado *Pedro* que está executando em uma máquina chamada *P1* terá `Pedro@P1:1099/JADE`[3] como seu nome global. Os endereços incluídos no *AID* são os endereços da plataforma onde o agente está executando. Estes endereços são usados quando um agente precisa comunicar-se com outros agentes de diferentes plataformas.

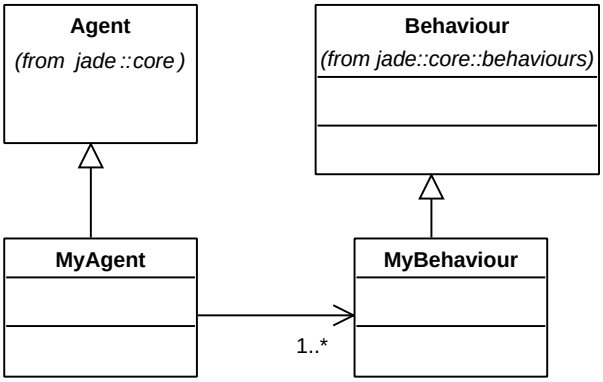


Figura 2.2: Agentes e seus comportamentos no JADE

2.2.3 Comunicação entre Agentes

A comunicação dos agentes em JADE é baseada em troca de mensagens, onde os agentes se comunicam criando e enviando mensagens individuais para outros agentes. As especificações de ACL (*Agent Communication Language*) da FIPA [12] padronizam o formato das mensagens no que diz respeito à codificação e à semântica das mensagens. Esta especificação não se envolve com o mecanismo de transporte de mensagens.

Uma mensagem ACL é representada por um objeto da classe *ACLMessage* do pacote *jade.lang.acl* que possui vários atributos:

- O agente remetente da mensagem;
- A lista de receptores;

[3] A porta 1099 é a porta padrão de uma plataforma JADE. A porta 1099 também é a porta padrão do RMI que o JADE utiliza para chamadas de métodos de objetos JAVA remotos.

- A intenção da comunicação (chamada de *"performative"*): indica o que o remetente pretende alcançar enviando a mensagem. A *performative* pode ser REQUEST, se o remetente quer que o receptor realize uma ação; INFORM, se o remetente quer avisar alguém sobre um fato; QUERY_IF, se pretende saber algo de acordo com uma condição informada; e CFP (*call for proporsal*), PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL, caso os participantes da comunicação encontrem-se em uma negociação;

- O conteúdo da mensagem (por exemplo, a ação a ser realizada em uma mensagem REQUEST ou o fato a ser informado em uma mensagem INFORM);

- A linguagem de conteúdo: representa a sintaxe usada para expressar o conteúdo da mensagem que deve ser concordada entre o remetente e o receptor da mensagem. Existem várias especificações de CL (*Content Language*) publicadas hoje pela FIPA[4];

- A ontologia: representa o vocabulário de símbolos usado no conteúdo da mensagem e seus significados. Uma ontologia deve ser concordada entre o remetente e o receptor da mensagem;

- Outros campos usados para controlar várias conversas concorrentes e para especificar *timeouts* de recebimento de resposta como identificador da conversação e outros.

Do ponto de vista do programador, o envio de uma mensagem ACL é feita simplemente pela chamada do método *send()* da classe que implementa o agente. Entretanto, do ponto de vista interno do JADE, três diferentes protocolos de transporte podem ser utilizados:

1. se o agente receptor estiver sendo executado no mesmo *container*, o objeto JAVA que representa a mensagem ACL é passada ao receptor usando eventos de objeto;

2. se o agente receptor estiver sendo executado na mesma plataforma mas em um container diferente, a mesma é transportada utilizando JAVA RMI[5]. O agente receptor recebe a mensagem ACL como uma entrega de mensagens "intra-container";

[4]As especificações de CL podem ser encontradas em *http://www.fipa.org/repository/cls.php3*.

[5]O JAVA RMI fornece mecanismos de chamada de métodos de objetos JAVA remotos e *marshaling* e *unmarshaling* transparente das mensagens, evitando conversões tediosas de mensagens

3. se o agente receptor estiver sendo executado em uma plataforma diferente, um MTP (Message Transport Protocol) é acionado para o transporte das mensagens.

Um MTP fornece mecanismos de transporte de mensagens entre agentes de diferentes plataformas. O trabalho do mecanismo de transporte do lado do remetente envolve traduzir um objeto de uma mensagem ACL em uma *string* de *bytes* obedecendo um padrão de codificação e transportar a mensagem para o agente receptor. O mecanismo de transporte do lado do receptor traduz a *string* de volta na mensagem ACL e a insere numa fila de mensagens até que o agente a resgate da fila e a processe, finalmente. A Figura 2.3 ilustra o funcionamento de um MTP JADE. Os mecanismos de transporte de mensagens são relativamente transparentes para o usuário.

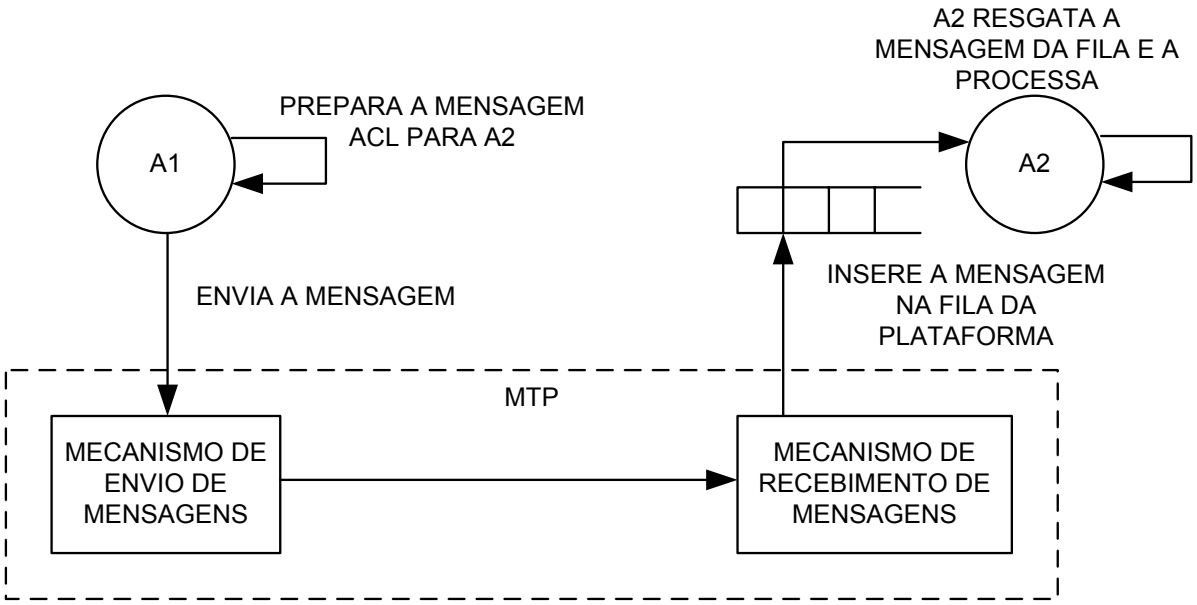


Figura 2.3: Funcionamento de um MTP JADE

2.3 Segurança

Segundo Tanenbaum [38], os problemas de segurança podem ser divididos nas seguintes áreas interligadas: confidenciabilidade (ou sigilo), autenticação, não-repúdio e integridade. A confidenciabilidade está relacionada ao fato de manter as informações longe de usuários não-autorizados. Em geral, a autenticação cuida do processo de determinar com quem você está se comunicando antes de revelar informações sigilosas ou entrar em uma transação comercial. O não-repúdio trata de assinaturas: como provar que seu cliente

fez realmente um pedido eletrônico de 10 milhões de unidades de um produto com preço unitário de 89 centavos quando mais tarde ele afirmar que o preço era 69 centavos? Ou então, é possível que afirme que nunca efetuou qualquer pedido. Por fim, A integridade trata de como você pode se certificar de que uma mensagem recebida é realmente legítima e não algo que um oponente mal-intencionado modificou ou inventou?

2.3.1 Princípios da Criptografia

De acordo com Tanenbaum [38], a confidenciabilidade se baseia em princípios criptográficos. A palavra criptografia significa "escrita secreta". Criptografia se resume no processo matemático, onde mensagens, conhecidas como texto simples (*plain text*), são transformadas por uma função que é parametrizada por uma chave. Em seguida, a saída do processo de criptografia, conhecida como texto cifrado (*cipher text*), é transmitida, normalmente através de um canal de comunicação. Admitimos que o inimigo, ou intruso, ouça e copie cuidadosamente o texto cifrado completo. No entanto, ao contrário do destinatário pretendido, ele não conhece a chave para descriptografar o texto e, portanto, não pode fazê-lo com muita facilidade. A ação de um intruso pode ser mais prejudicial do que somente escutar o canal de comunicação, ele também pode capturar mensagens e reproduzí-las, mais tarde, injetando suas próprias mensagens ou modificar mensagens legítimas antes que elas cheguem ao destinatário.

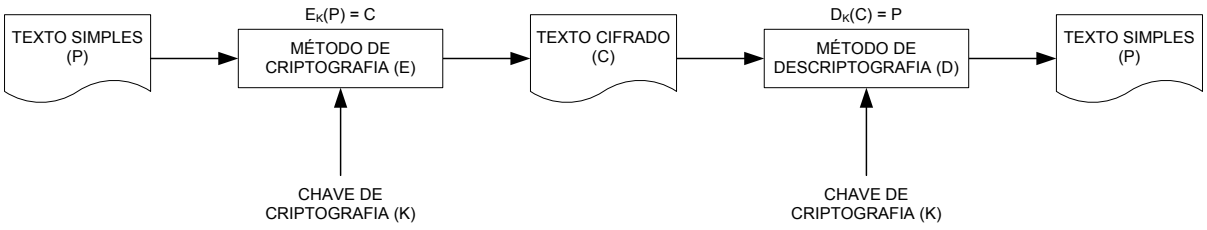


Figura 2.4: Os princípios da criptografia

Com freqüência, será útil e prático ter uma notação para estabelecer uma relação entre o texto simples, o texto cifrado e as chaves. Utilizaremos $C = E_K(P)$ para denotar que a criptografia do texto simples P usando a chave K gera o texto cifrado C. Da mesma forma, $P = D_K(C)$ representa a descriptografia de C para se obter o texto simples outra vez. Então, temos [38]:

$$D_K(E_K(P)) = P.$$

Essa notação sugere que E e D são simplesmente funções matemáticas, o que é verdade. A única parte complicada é que ambas são funções de dois parâmetros, e escrevemos um desses parâmetros (a chave) como um caractere subscrito, em vez de representá-lo como um argumento, para distinguí-lo da mensagem. Os princípios da criptografia estão representados na Figura 2.4.

Uma chave de criptografia consiste em uma *string* de *bits* que representa uma das muitas formas possíveis de criptografia. Ao contrário do algoritmo do método de criptografia e descriptografia, que só pode ser modificado a cada período de alguns anos, quando alguma (quando da alteração da versão do algoritmo), a chave pode ser alterada sempre que necessário. Inclusive, o algoritmo de criptografia pode ser representado por um método publicamente conhecido, parametrizado por uma chave secreta que pode ser alterada com facilidade. A idéia de que os algoritmos de criptografia sejam públicos e que o segredo reside exclusivamente nas chave é chamado de princípio Kerckhoff, que recebeu esse nome em homenagem ao criptógrafo militar flamengo Auguste Kerckhoff que o anunciou primeiro em 1883. O princípio diz que:

Todos os algoritmos devem ser públicos; *apenas as chaves são secretas.*

O caráter não-sigiloso do algoritmo deve ser sempre obedecido. Tentar manter o algoritmo secreto, uma estratégia conhecida no ramo como segurança pela obscuridade, nunca funciona.

Na verdade, o sigilo está na chave, e seu tamanho é uma questão muito importante do projeto. Quanto maior for a chave, mais alto será o fator de trabalho que alguém engajado em quebrar o seu texto cifrado terá de lidar. O fator de trabalho para decodificar o sistema através de uma exaustiva pesquisa no espaço da chave é exponencial em relação ao tamanho da chave. O sigilo é decorrente da presença de um algoritmo forte (mas público) e de uma chave longa.

Algoritmos de criptografia que utilizam a mesma chave para o processo de criptografia e descriptografia de mensagens são chamados de algoritmos de chave simétrica. Algoritmos de criptografia que utilizam uma chave para o processo de criptografia e outra chave para o processo de descriptografia da mesma mensagem são chamados de algoritmos de chave assimétrica.

2.3.2 Algoritmo DES e 3DES

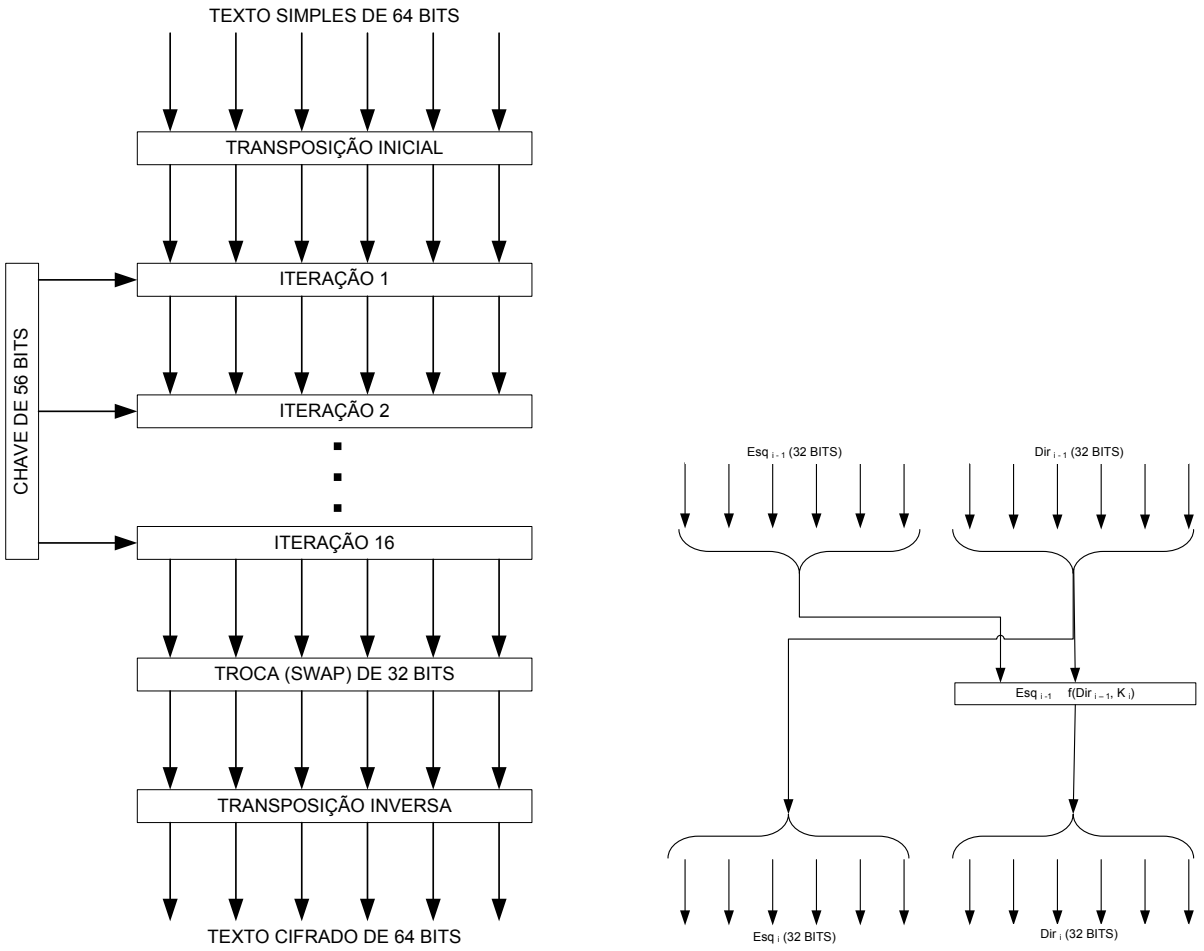
O algoritmo DES (*Data Encryption Standard*) foi desenvolvido pela IBM e adotada pelo governo dos Estados Unidos em janeiro de 1977 [38]. O algoritmo DES foi amplamente adotado pelo setor de informática para o uso em produtos de segurança. Em sua forma original, este algoritmo já não é mais seguro; no entanto, em uma forma modificada, ainda é útil.

Resumidamente, o processo de criptografia do algoritmo DES produz vários blocos de 64 *bits* cifrados a partir do texto original, que são submetidos a vários estágios de permuta das posições dos *bits*. O algoritmo DES possui 19 estágios de permutas de *bits*, 16 deles são parametrizados por uma chave de 56 *bits*. O primeiro deles é uma transposição independente da chave no texto simples de 64 *bits*. O último estágio é exatamente o inverso dessa transposição. O penúltimo estágio troca os 32 *bits* mais à esquerda pelos 32 *bits* mais à direita. Os 16 restantes são funcionalmente idênticos, mas são parametrizados por diferentes funções da chave. O algoritmo de criptografia DES está representado na Figura 2.5 (a).

O processo de descriptografia se dá simplesmente executando o mesmo procedimento de forma inversa, claro, utilizando a mesma chave.

A operação desses estágios intermediários é ilustrada na Figura 2.5 (b). Cada estágio utiliza duas entradas de 32 *bits* e produz duas saídas de 32 *bits*. A saída da esquerda é apenas uma cópia da entrada da direita. A saída da direita é formada pelo resultado do OR exclusivo *bit* a *bit* aplicado à entrada da esquerda e uma função de entrada da direita com a chave desse estágio, K_i. Toda a complexidade reside nessa função. Em cada uma das 16 iterações, é utilizada uma chave diferente. Antes de se iniciar o algoritmo, é aplicada à chave uma transposição de 56 *bits*. Antes de cada iteração, a chave é particionada em duas unidades de 28 bits, sendo cada uma delas girada a esquerda um número de *bits* que depende do número da iteração. K_i é derivada dessa chave girada pela aplicação de mais uma transposição de 56 bits sobre ela. Em cada rodada, um subconjunto de 48 *bits* dos 56 *bits* é extraído e permutado.

No início de 1979, a IBM percebeu que o tamanho da mensagem DES era muito pequeno e criou uma forma de aumentá-lo aplicando o mesmo processo de criptografia repetido três vezes com chaves diferentes, criando assim o algoritmo 3DES ou triplo DES.



(a) (b)

Figura 2.5: Criptografia DES

Uma das formas de se utilizar o algoritmo 3DES consiste em criptografar a mensagem original com uma chave, e descriptografar o produto cifrado com outra chave. Descriptografar um código cifrado gera outro código cifrado. A saída então é novamente criptografada com a primeira chave, gerando outra mensagem cifrada. O processo de descriptografia do primeiro código cifrado com outra chave resulta em outra mensagem cifrada e "economiza" a utilização de uma terceira chave. A Figura 2.6 representa um processo de criptografia 3DES de um texto simples P e descriptografia do texto cifrado C utilizando somente duas chaves (K_1 e K_2).

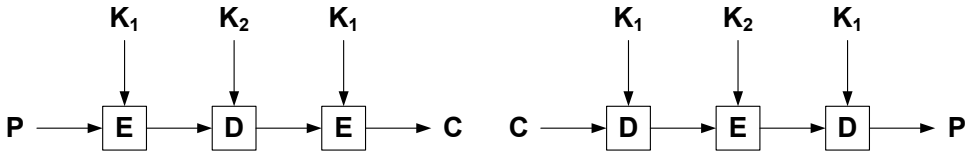


Figura 2.6: Criptografia e descriptografia 3DES com duas chaves

2.3.3 Algoritmo RSA

Segundo Tanenbaum [38], historicamente, o problema da distribuição de chaves sempre foi o elo mais fraco da maioria dos sistemas de criptografia. Independentemente de quanto um sistema de criptografia fosse sólido, se um intruso conseguisse roubar a chave, o sistema acabava sendo inútil.

A abordagem de criptografia de chave simétrica possui então este problema inerente: as chaves devem ser protegidas contra roubos, mas devem ser também distribuídas aos participantes da comunicação.

A abordagem de chave assimétrica funciona da seguinte maneira: uma pessoa que deseja receber mensagens secretas, primeiro cria duas chaves. Uma das chaves se torna pública e a outra é mantida em segredo. Sempre que alguém necessitar enviar uma mensagem secreta, deve criptografar a mensagem utilizando a chave pública. Uma mensagem criptografada por uma chave pública somente pode ser descriptografada pela chave privada, por isso o algoritmo também é chamado de criptografia de chave pública.

Utilizando a abordagem de chaves públicas e privadas, o algoritmo de chaves assimétricas resolve a questão da distribuição e da proteção observada no algoritmo de chaves simétricas.

O RSA[6] corresponde a um algoritmo de chaves assimétricas. O RSA sobreviveu a todas as tentativas de rompimento por muito tempo e é considerado um algoritmo muito forte. Porém, O RSA requer chaves muito pesada e grandes.

O Funcionamento do RSA consiste em:

1. escolher dois números primos extensos, p e q (geralmente de 1024 *bits*);
2. calcular $n = p \times q$ e $z = (p - 1) \times (q - 1)$;
3. escolher um número d tal que z e d sejam primos entre si; e
4. encontrar e de forma que $e \times d = 1 \; mod \; z$.

Com esses parâmetros calculados com antecedência, estamos prontos para começar a criptografia. Divida o texto simples (considerado uma *string* de *bits*) em blocos, de modo que cada mensagem de texto simples P fique no intervalo $0 \leq P < n$.

[6] O RSA foi batizado assim pelas iniciais dos três criadores do algoritmo: Rivest, Shamir e Adelman.

Isso pode ser feito agrupando-se o texto simples em blocos de k *bits*, onde k é o maior inteiro para o qual a desigualdade $2^k < n$ é verdadeira.

Para criptografar a mensagem P, calcule $C = P^e (mod\, n)$. Para descriptografar C, calcule $P = C^d (mod\ n)$. É possível provar, que para todo P na faixa especificada, as funções de criptografia e descriptografia são inversas entre si. Para realizar a criptografia, você precisa de e e n, enquanto para a descriptografia são necessários d e n. Portanto, a chave pública consiste no par (e, n) e a chave privada no par (d, n).

Se pudesse fatorar o valor n (publicamente conhecido), poderíamos então encontrar o p e q e, a partir destes, encontrar z. Com o conhecimento de z e e, o valor de d pode ser encontrado utilizando-se o algoritmo de Euclides. Felizmente, os matemáticos têm tentado fatorar números extensos por pelo menos 300 anos, e o conhecimento acumulado sugere que o problema é extremamente difícil.

De acordo com Rivest e seus colegas, a fatoração de um número de 500 digítos requer 1025 anos, usando a força bruta e pressupondo que se está usando o melhor algoritmo conhecido e um computador com tempo por instrução de 1 μs.

Um exemplo didático muito comum do algoritmo RSA [38] é mostrado na Figura 2.7. Para este exemplo, escolhemos $p = 3$ e $q = 11$, o que gera $n = 33$ e $z = 20$. Um valor adequado para d é $d = 7$, visto que 7 e 20 não têm fatores comuns. Com esses parâmetros, e pode ser encontrado resolvendo a equação $7e = 1(mod 20)$, que produz $e = 3$. O texto cifrado C para uma mensagem de texto simples P é dado por $C = P^3 (mod 33)$. O texto cifrado é decodificado pelo receptor usando a regra $P = C^7 (mod 33)$. A Figura 2.7 mostra a codificação do texto simples "SUZANNE" como exemplo.

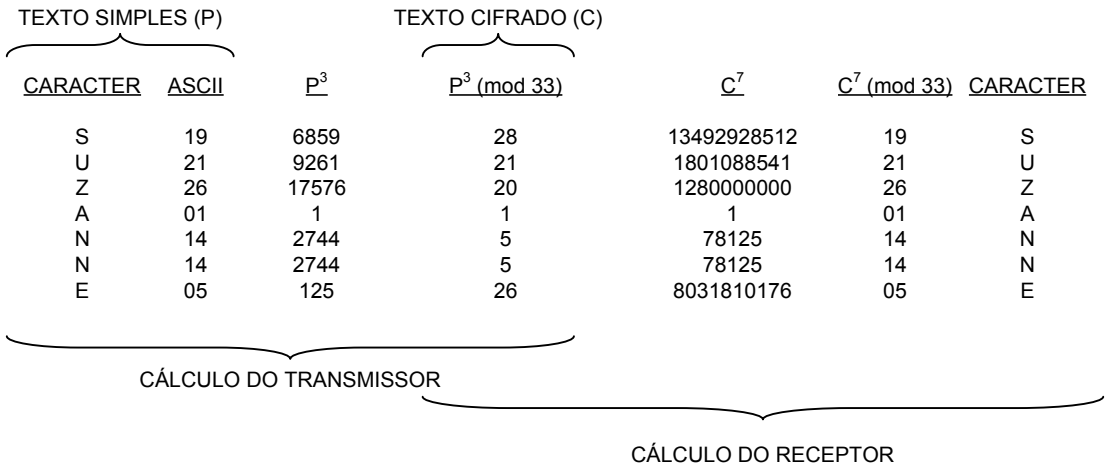


Figura 2.7: Exemplo da criptografia RSA

O RSA baseia sua segurança e implementação em vários princípios matemáticos e na dificuldade de fatorar números extensos. Por possuir chaves grandes e pesadas e se apresentar como um algoritmo lento, o RSA é utilizada para distribuir chaves secretas usadas em sessões, e que por sua vez, são usadas em algoritmos de chave simétrica, como o 3DES.

2.3.4 Algoritmo *Diffie-Hellman*

O Algoritmo *Diffie-Hellman* [38], também chamado de troca de chaves de *Diffie-Hellman* corresponde a um protocolo que permite o estabelecimento de uma chave secreta entre computadores. O algoritmo funciona da seguinte maneira: Alice e Bob têm de concordar em relação a dois números extensos, n e g, onde n é um número primo, $(n-1)/2$ também é um número primo e onde certas condições se aplicam a g. Esses números podem ser públicos; portanto, um deles só precisa selecionar n e g e informar ao outro abertamente. Agora, Alice escolhe um grande número x (digamos, de 512 *bits*) e o mantém em segredo. Da mesma forma, Bob seleciona um grande número secreto, y.

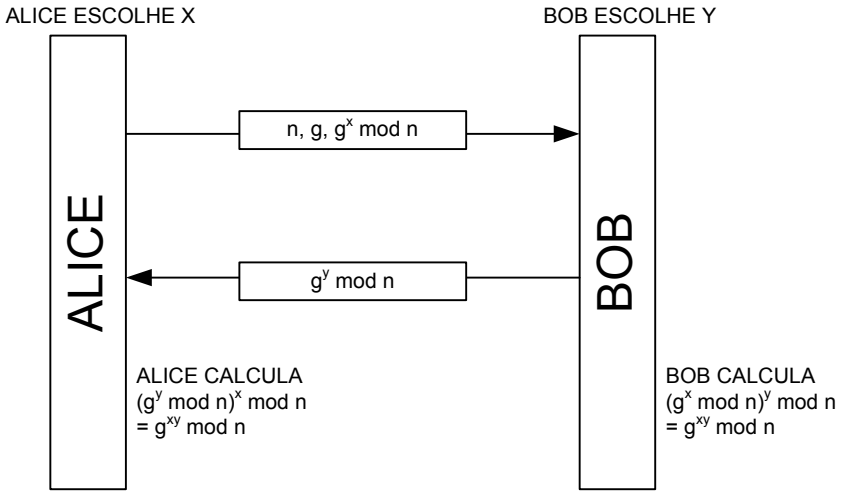


Figura 2.8: Algoritmo *Diffie-Hellman*

Alice inicia um protocolo de troca de chaves enviando a Bob uma mensagem contendo n, g e $g^x mod n$. Bob responde enviando a Alice uma mensagem contendo $g^y mod n$. Agora, Alice eleva a x-ésima potência em módulo n o número que Bob lhe enviou, a fim de obter $(g^y mod n)^x mod n$. Bob executa uma operação semelhante para obter $(g^x mod n)^y mod n$. Pelas leis da aritmética modular, ambos os cálculos produzem $g^{xy} mod n$. Agora, Alice e Bob compartilham uma chave secreta, $g^{xy} mod n$.

Um intruso, que possa escutar a comunicação entre Alice e Bob, poderia interceptar os valores transmitidos por Alice. Se este intruso, conseguir capturar g e n e calcular x e y, ele poderia obter a chave secreta compartilhada entre Alice e Bob. A vantagem deste algoritmo é que com somente $g^x mod n$, não é possível calcular x. Não existem algoritmos práticos para o cálculo de logaritmos discretos cuja base seja um número primo muito extenso conhecido.

A Figura 2.8 ilustra o algoritmo: definindo $n = 47$ e $g = 3$, Alice poderia escolher $x = 8$ (passo 1 na figura) e Bob escolher $y = 10$ (passo 2). A mensagem de Alice para Bob é $(47, 3, 28)$ (passo 3), pois 3^8 *mod* 47 é igual a 28. A mensagem de Bob para Alice é (17) (passo 4). Alice calcula 17^8 *mod* 47, que é igual a 4. Bob calcula 28^{10} *mod* 47, que é igual a 4. Alice e Bob determinam de forma independente que agora a chave secreta é 4. Um intruso tem de resolver a equação 3^x *mod* 47 = 28, o que pode ser feito por pesquisa exaustiva de números pequenos como esse, mas não quando todos os números têm centenas de *bits*. Todos os algoritmos conhecidos até o momento demoram muito tempo para realizar esse cálculo, mesmo em supercomputadores maciçamente paralelos [38].

Apesar da elegância do algoritmo de *Diffie-Hellman*, há um problema: quando Bob obtém a mensagem de Alice, não há certeza de que esta mensagem veio realmente de Alice e um intruso pode explorar este fato para enganar Alice e Bob, como ilustra a Figura 2.9. Aqui, enquanto, Alice e Bob estão escolhendo x e y, respectivamente, um intruso escolhe seu próprio número aleatório, z. Alice envia a mensagem 1 para Bob. O intruso a intercepta e envia a mensagem 2 para Bob, usando g e n corretos, mas com o seu próprio z em vez de x. Bob envia a mensagem 3 para Alice, que é também interceptada pelo intruso e faz com que Alice receba a mensagem 4.

Agora, todos utilizam a aritmética modular. Alice calcula a chave secreta como $g^{xz} mod n$, e o intruso faz o mesmo. Bob calcula $g^{yz} mod n$ e o intruso faz o mesmo. Alice pensa que está comunicando com Bob, e portanto, estabelece uma chave de sessão com o intruso. Bob faz o mesmo. Todas as mensagens que Alice envia na sessão são criptografadas e capturadas pelo intruso que consegue descriptografá-las, lê-las e criptografá-las novamente utilizando a chave de sessão estabelecida entre ele e Bob. O mesmo acontece com as mensagens transmitidas de Bob para Alice. Este ataque é chamado de ataque da brigada de incêndio (pois lembra vagamente um antigo corpo de bombeiros formado por voluntários que, enfileirados, passavam baldes d'água de mão

em mão do caminhão de água até o incêndio) ou ataque do Homem-do-Meio (*Man-In-The-Middle Attack*). A Figura 2.9 representa o ataque do Homem-do-Meio aplicado ao *Diffie-Hellman*.

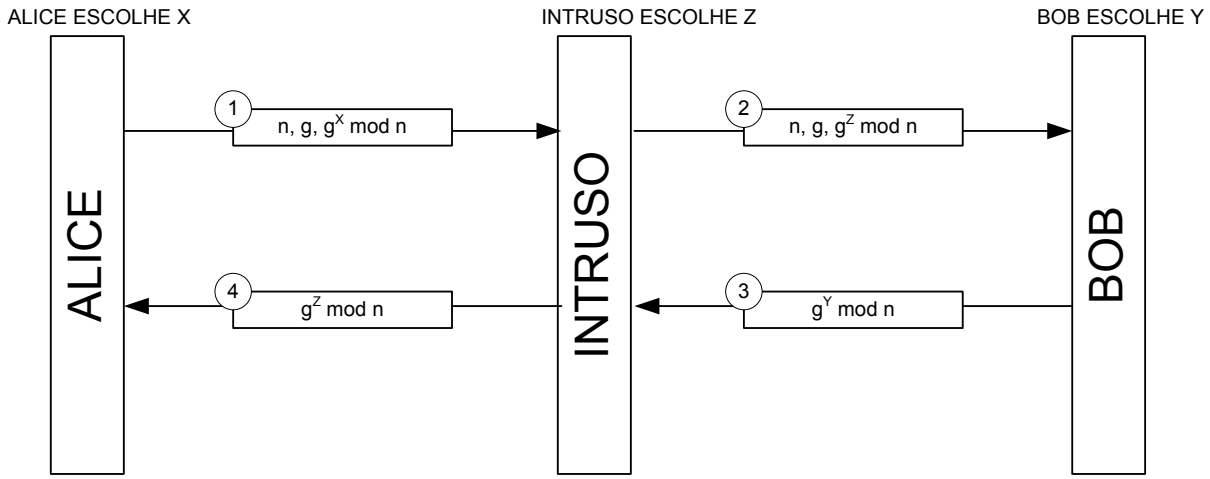


Figura 2.9: Ataque do Homem-do-Meio aplicado ao *Diffie-Hellman*

2.3.5 Assinatura Digital

A autenticidade de muitos documentos legais, financeiros e outros documentos é determinada pela presença de uma assinatura autorizada. Para que os sistemas de mensagens computadorizadas possam substituir o transporte físico de documentos em papel e tinta, deve-se encontrar um método que permita assinar os documentos de um modo que não possa ser forjado [38].

O problema de se criar um substituto para as assinaturas escritas à mão é muito difícil. Basicamente, necessita-se de um sistema através do qual uma parte possa enviar uma mensagem "assinada" para outra parte de forma que:

1. o receptor possa verificar a identidade alegada pelo transmissor;
2. posteriormente, o transmissor não possa repudiar o conteúdo da mensagem;
3. o receptor não tenha a possibilidade de forjar ele mesmo a mensagem.

O primeiro requisito é necessário, por exemplo, em sistemas financeiros. Quando um cliente solicita um pedido a um banco via *Internet banking* para comprar uma tonelada de ouro, o banco precisa se assegurar que o computador que está originando o

pedido realmente pertence à pessoa possuidora da conta que está sendo debitada. Sendo assim, o banco tem que autenticar o cliente e o cliente deve autenticar o banco também.

O segundo requisito é necessário para proteger o banco contra fraudes. Supomos que o banco compre a tonelada de ouro, e imediatamente após isso o preço do ouro caia violentamente. Um cliente desonesto diante da perda de dinheiro pode reclamar que nunca realizou qualquer pedido ao banco. A propriedade que nenhuma parte de um contrato possa negar que realmente tenha assinado o contrato se chama não-repúdio.

O terceiro requisito é necessário para proteger o cliente caso o preço do ouro dobre após a compra do outro pelo banco. O banco agora de posse de uma fortuna pode dizer que o cliente somente solicitou uma barra de ouro em vez de uma tonelada.

A criptografia de chave pública (ou de chave assimétrica) traz uma grande contribuição para o processo de assinatura digital. Já sabemos que os algoritmos de criptografia e descriptografia de chave pública possuem a propriedade $D(C(P)) = P$, ou seja, sabemos que o produto da criptografia de um texto qualquer utilizando uma chave pública pode ser submetido ao algoritmo de descriptografia utilizando a chave privada correspondente e resultar na volta do texto original.

O que é apresentado de novo agora é que a propriedade $C(D(P)) = P$ também é verdadeira. Em outras palavras, se submetermos um texto qualquer ao processo de descriptografia utilizando uma chave privada resulta em um produto que pode ser submetido ao algoritmo de criptografia utilizando a chave pública correspondente e para produzir o texto original novamente. Assim, podemos utilizar chaves privadas e públicas para assinar e verificar digitalmente um documento.

A Figura 2.10 ilustra o processo de assinatura e verificação digital utilizando chaves públicas e privada [38]. Na figura, Alice envia uma mensagem assinada, P, para Bob transmitindo a mensagem $C_B(D_A(P))$, onde D_A representa a descriptografia da mensagem original utilizando a chave privada de Alice e C_B representa a criptografia utilizando a chave pública de Bob. Assim, Alice antes de criptografar a mensagem para Bob utilizando a chave pública de Bob, submete a mensagem original ao algoritmo de descriptografia utilizando a sua própria chave privada.

Quando Bob recebe a mensagem, ele a descriptografa utilizando sua chave privada (D_B) para obter o produto da assinatura digital de Alice. De posse do produto da assinatura, Bob pode aplicar a criptografia (E_A) para recuperar a mensagem original.

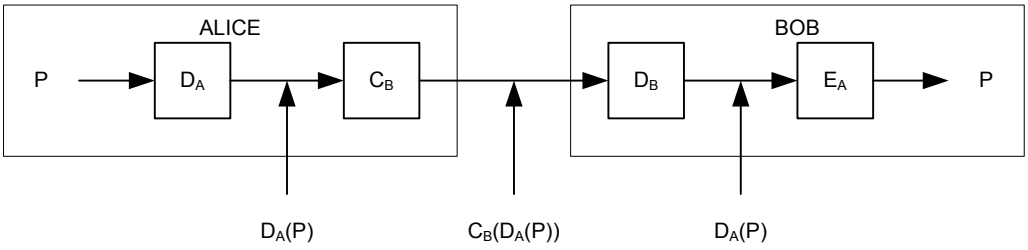


Figura 2.10: Assinatura digital com chaves assimétricas

Ao assinar uma mensagem é gerado um sumário de mensagem. Um sumário de mensagem corresponde à saída de uma função unidirecional que utiliza o texto da mensagem como parâmetro.

Um problema quanto ao esquema de assinatura digital mostrado é que ele oferece autenticação junto com a privacidade. Entretanto, existem cenários onde a autenticação é necessária, mas a privacidade não é. Para estes cenários são utilizados esquemas de sumários de mensagem (*message digest*).

Um esquema de sumário de mensagem é baseado na idéia que uma função de *hash* unidirecional pode produzir uma *string* de *bits* de tamanho fixo a partir de uma mensagem P. Esta função que pode ser chamada de MD possui quatro importantes propriedades [38]:

1. dado P, é fácil calcular $MD(P)$;
2. dado $MD(P)$, é praticamente impossível encontrar P;
3. dado P é impossível calcular um P' que satisfaça $MD(P') = MD(P)$;
4. a mudança de mesmo 1 *bit* em P produz um $MD(P)$ muito diferente.

Para satisfazer a propriedade 3, o *hash* deve ter pelo menos 128 *bits*. Para satisfazer a propriedade 4, o *hash* deve embaralhar bastante os *bits*.

Em um cenário de criptografia de chave pública, os sumários de mensagem podem ser utilizados junto com a chave privada para fornecer autenticidade, integridade e não-repúdio. Na Figura 2.11, Alice primeiro calcula o *hash* de sua mensagem P e, depois, submete o *hash* ao algoritmo de descriptografia D_A utilizando sua chave privada. Alice ao enviar a mensagem a Bob anexa o produto $D_A(MD(P))$. Bob ao receber a mensagem submete o anexo da mensagem $D_A(MD(P))$ ao seu algoritmo de criptografia utilizando

a chave pública de Alice para obter o $MD(P)$ enviado por Alice novamente. Bob gera novamente o $MD(P)$ e compara com o que foi recebido, se ambos forem iguais então a mensagem foi realmente enviada por Alice e a mensagem não sofre nenhuma alteração durante o transporte.

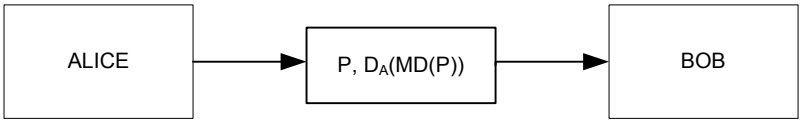


Figura 2.11: Assinatura digital com somente autenticidade e integridade

2.3.6 Esquema PKI

Como uma primeira tentativa de distribuição de chaves públicas com segurança, poderíamos imaginar um centro de distribuição de chaves disponível *on-line* 24 horas por dia a fim de fornecer chaves públicas por demanda. Um dos muitos problemas com essa solução é o fato de ela não ser escalável, e o centro de distribuição de chaves rapidamente se tornaria um gargalo. Além disso, se o tal centro de distribuição de chaves públicas ficasse inativo, a segurança da Internet seria paralisada repentinamente.

Por essas razões, foi desenvolvido uma solução diferente que não exige que o centro de distribuição de chaves esteja *on-line* todo o tempo. De fato, o centro de distribuição não precisa estar *on-line* de modo algum. Em vez disso, ele certifica as chaves públicas pertencentes a pessoas, empresas e outras organizações. Uma organização que certifica chaves públicas é chamada CA (*Certification Authority*).

Como um exemplo, suponha que Bob queira permitir que Alice e outras pessoas se comuniquem com ele em segurança. Ele pode ir até a CA com sua chave pública e seu passaporte ou com a carteira de motorista e solicitar a certificação. A CA emite então um certificado semelhante a Figura 2.12 e assina o certificado gerando um *hash* do próprio certificado e sua chave privada.

A principal função de um certificado é vincular uma chave pública ao nome de um protagonista (indivíduo, empresa, etc). Os certificados em si não são secretos ou protegidos. Por exemplo, Bob poderia decidir colocar seu novo certificado em seu *Web Site*, com um *link* na página principal informando: clique aqui para ver meu certificado de chave pública. O clique resultante retornaria o certificado e o bloco de assinatura.

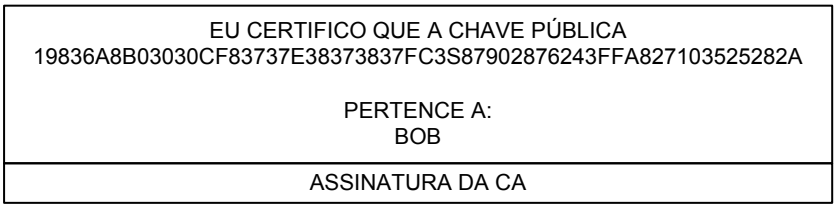


Figura 2.12: Exemplo de um certificado

Fazer uma única CA emitir todos os certificados do mundo evidentemente não funcionaria. Ela entraria em colapso sob a carga e também seria um ponto central de falha. Uma solução possível poderia ser a existência de várias CAs, todas administradas pela mesma organização e todas usando a mesma chave privada para assinar certificados. Embora, isso pudesse resolver o problema da carga e da falha, há um novo problema: o vazamento de chaves. Se houvesse dezenas de servidores espalhados pelo mundo, todos com a chave privada da CA, a chance de que a chave privada fosse roubada ou sofresse algum tipo de vazamento seria bastante alta. E ainda há o problema de aceitação da organização que operaria a CA, afinal, é impossível encontrar uma organização que fosse aceita em todo o mundo com uma entidade legítima e confiável.

Por essas razões, foi desenvolvido um modo diferente de certificar chaves públicas, identificada pelo nome geral PKI (*Public Key Infrastrucure*) [38].

Uma PKI tem vários componentes, incluindo usuários, CAs, certificados e diretórios. A função da PKI é fornecer um modo de estruturar esses componentes e definir padrões para os vários documentos e protocolos. Uma forma particularmente simples de PKI é uma hierarquia de CAs, como mostra a Figura 2.13. Neste exemplo, mostramos três níveis, mas, na prática, pode haver um número menor ou maior. A CA de nível superior, chamada raiz, certifica CAs do segundo nível, que denominamos RAs (*Regional Authorities*), porque podem cobrir alguma região geográfica, como um país ou um continente. Entretanto, esse termo não é padrão; de fato, nenhum termo é realmente padrão para os diferentes níveis da árvore. Por sua vez, as RAs certificam as CAs reais, que emitem certificados X.509[7] para organizações e indivíduos. Quando a raiz autoriza uma nova RA, ela gera um certificado X.509 anunciando que aprovou a RA, inclui a chave pública da nova RA no certificado, assina o certificado e o entrega à RA. De modo semelhante, quando uma RA aprova uma nova CA, ela produz e assina um certificado declarando sua aprovação e contendo a chave pública da CA.

[7]O X.509 consiste em um formato de padrão para certificados bastante utilizado na Internet.

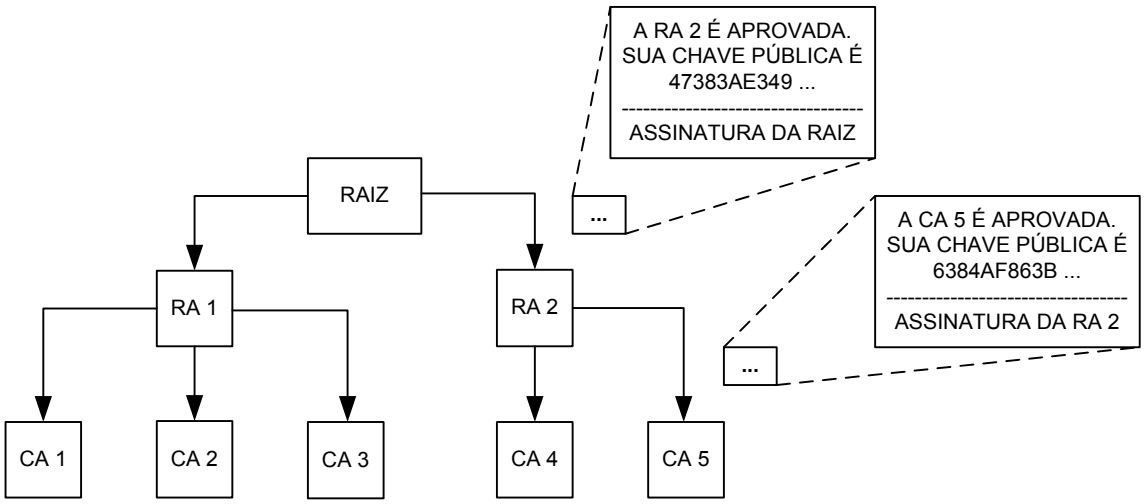


Figura 2.13: Esquema PKI

Nossa PKI funciona de modo semelhante. Suponha que Alice precise da chave pública de Bob, a fim de se comunicar com ele; então, ela procura e encontra um certificado contendo a chave assinada pela CA 5. Ela poderia ir até a CA 5 e pedir por sua legitimidade. A CA 5 responde com o certificado que recebeu da RA 2, que contém a chave pública da CA 5. Agora, munida da chave pública da CA 5, Alice pode confirmar que o certificado de Bob foi de fato assinado pela CA 5 e, portanto, é válido. A próxima etapa é pedir a RA 2 que prove sua legitimidade. A resposta à consulta de Alice é um certificado assinado pela raiz e contendo a chave pública de RA 2. Agora, Alice tem certeza que possui a chave pública de Bob.

Supõe-se que todo mundo conhece a chave pública da raiz. Por exemplo, seu navegador pode ter sido comercializado com a chave pública da raiz embutida.

Como Bob sabe que Alice precisa verificar a CA 5 e a RA 2, ele reúne os dois certificados necessários e os fornece a ela juntamente com o seu próprio certificado. Agora, ela pode usar seu conhecimento da chave pública da raiz para confirmar os certificados de nível superior e a chave pública que contém para verificar o segundo certificado. Desse modo, Alice não precisa entrar em contato com ninguém para fazer a verificação. Como os certificados são todos assinados, ela pode detectar com facilidade quaisquer tentativas de falsificar seu conteúdo. Uma cadeia de certificados como essa que volta à raiz, às vezes é chamada cadeia de confiança ou caminho de certificação. A técnica é amplamente utilizada na prática.

É claro que ainda temos o problema de saber quem vai administrar a raiz. A

solução é não ter uma única raiz, mas sim várias raízes, cada uma com suas próprias RAs e CAs. De fato, os navegadores modernos são pré-carregados com as chaves públicas de mais de 100 raízes, às vezes referidas como âncoras de confiança. Desse modo, pode-se evitar ter uma única autoridade confiável no mundo inteiro.

Entretanto, agora existe a questão de como o fornecedor do navegador decide quais das suposta âncoras de confiança são de fato confiáveis e quais são desprezíveis. Tudo se reduz à confiança do usuário no fornecedor do navegador para fazer escolhas sensatas e não aprovar simplesmente todas as âncoras de confiança dispostas a pagar por sua inclusão na lista. A maioria dos navegadores permite que os usuários inspecionem as chaves da raiz (sob a forma de certificados assinados pela raiz) e eliminem qualquer uma que pareça obscura.

2.4 Conclusões

Neste capítulo, nós apresentamos os conceitos básicos e a visão geral dos sistemas multiagentes necessários para a compreensão deste trabalho.

Nós apresentamos o *framework* JADE que é utilizado para a implementação dos protótipos das soluções propostas neste trabalho. Do *framework* JADE, nós apresentamos o ambiente de execução dos agentes em JADE; os conceitos de *containers* e plataformas; a implementação de um agente em JADE; a identificação de um agente em JADE; os conceitos de comunicação entre agentes em JADE e o funcionamento de um MTP-JADE.

Nós apresentamos também todos os algoritmos e esquemas de segurança que serão referenciados no restante do trabalho. Dos conceitos de segurança, nós apresentamos os conceitos de confidenciabilidade, integridade, não-repúdio e autenticação; os princípios da criptografia e os principais algoritmos; o algoritmo *Diffie-Hellman* de chave secreta; os princípios da assinatura digital e seus principais algoritmos; o esquema *PKI*.

3 Soluções Existentes

Já que XML pode ser utilizada para representar mensagens, a segurança XML tem seu foco na segurança de mensagens. Criptografia e assinatura digital de mensagens são as principais técnicas utilizadas. Embora técnicas de criptografia de arquivos ou de *e-mails* possam ser usadas para criptografar documentos XML, as técnicas de criptografia baseadas em XML são mais adaptadas para criptografar documentos XML e mais adaptadas para a *Web*. Atualmente, a W3C [1] [40] lidera a maioria dos esforços de padronização de segurança XML.

Neste capítulo, nós apresentamos em um nível mais detalhado as especificações XML que serão adaptadas nas soluções de segurança e de confiabilidade. O padrão RDF também é apresentado. Neste capítulo, nós também apresentaremos as soluções de segurança e de confiabilidade existentes para o JADE.

3.1 Especificação *XML Encryption*

A recomendação candidata da W3C *XML Encryption Syntax and Processing* [9], [17] define um processo de criptografia digital de dados e o método com o que o resultado da criptografia de dados deve ser representado em XML. Embora os dados a serem criptografados possam ser representados em uma forma mais genérica do que XML, dados em XML estão em melhores condições para o *XML Encryption*. A especificação *XML Encryption* (ou somente XML-Enc) suporta a criptografia de todo um documento XML ou de partes definidas dele. A menor unidade de informação que pode ser criptografada é um elemento.

A recomendação W3C foca na definição do processo para a criação e para a representação dos dados criptografados e codificados em XML. A recomendação também aborda o processo de criptografia, mas não tenta definir novos algoritmos. Ela especifica algoritmos existentes para o processo de criptografia e descriptografia, acordo de chaves,

[1] A W3C *World Wide Web Consortium* é um consórcio internacional de companhias envolvidas com a Internet. O propósito da organização é desenvolver padrões abertos voltados para a Internet.

sumários de mensagens, autenticação de mensagens e outras aplicações de criptografia (exceto por assinaturas digitais, que é abordada em uma especificação diferente, a XML-DSig). Ela também aborda criptografia assimétrica e simétrica.

A especificação XML-Enc não abriga ao uso de algoritmos particulares ou tamanhos de chave. É responsabilidade do usuário assegurar-se de que as escolhas corretas estão sendo feitas. O implementador do sistema deve cuidadosamente decidir pelos algoritmos utilizados e pelo tamanho das chaves utilizados.

3.1.1 Formato e Estrutura da XML-Enc

A especificação XML-Enc possui um elemento *CipherData* que consiste resumidamente nos dados criptografados. O elemento *CipherData* pode ser representado de duas formas. A primeira forma é ser posto no documento XML, que é o jeito mais utilizado. Os dados criptografados que já não são mais humanamente compreensíveis como texto, pois são codificados em *base64*[2]. A segunda forma é que o *CipherData* refere-se ao objeto criptografado.

O elemento *EncryptionMethod* contém informações sobre o algoritmo de criptografia e o tamanho da chave.

O elemento *KeyInfo* fornece a informação necessária para que a aplicação do receptor possa descriptografar a informação cifrada. Se o elemento *KeyInfo* for omitido, espera-se que a aplicação receptora já saiba como realizar a descriptografia, incluindo qual chave usar. A XML-Enc suporta várias opções de especificação de chaves. Isto inclui um identificador de chaves, a chave de descriptografia, uma referência para a localização da chave de descriptografia, ou o certificado da chave pública que foi usada para a criptografia dos dados. Vários formatos de representação de certificados são utilizados, inclusive, o X.509. Finalmente, *EncryptionProperties* possui informações adicionais referentes a criptografia.

[2]A codificação *Base64* usa um conjunto de 64 caracteres (A-Za-z0-9+/=) para a representação de dados binários.

3.1.2 Processo de Criptografia e Descriptografia da XML-Enc

A especificação XML-Enc aconselha o seguinte procedimento para criptografar mensagens:

1. Escolher o algoritmo de criptografia e os parâmetros;
2. Obter a chave. Se a chave deve ser identificada, construí-se um elemento *KeyInfo*. Se a chave for encaminhada junto com a mensagem, deve-se criptografar a chave, construir um elemento *EncryptedKey* e colocá-lo em um elemento *KeyInfo* ou em outra porção do documento;
3. Criptografar os dados. Para dados XML, isto pode invocar a transformação UTF-8[3] para codificação e serialização. O resultado é uma *string* de octetos;
4. Construir a estrutura *EncryptedType*. Onde os dados criptografados é atualmente armazenados na estrutura, ao invés de serem referenciados, os dados criptografados devem ser codificados na base64. Contruir a estrutura *EncryptedType*, em outras palavras, significa construir uma estrutura *EncryptedData* ou uma estrutura *EncryptedKey*;
5. Substitua o elemento criptografado do documento XML com a estrutura *EncryptedType*.

A XML-Enc aconselha o seguinte procedimento para decriptografar uma mensagem:

1. Processar a estrutura *EncryptedType*.
2. Obter a chave de descriptografia;
3. Descriptografar os dados contidos no elemento *CipherData*;
4. Processar os dados criptografados. Isto requer que a aplicação substitua os dados descriptografados, que estão em formato UTF-8 para sua forma original. Isto deve

[3]O formato de transmissão *Unicode 8* (ou somente UTF-8) é um conjunto de caracteres recomendado para protocolos que vão além do uso do ASCII. O UTF-8 fornece suporte aos caracteres ASCII estendidos e à conversão do UCS-2, um conjunto de caracteres Unicode de 16 bits internacional. Permite um intervalo de nomes maior do que pode ser obtido com o uso de codificação ASCII para dados de caracteres.

ser capaz de substituir a estrutura *CipherData* no documento XML com o resultado da descriptografia.

3.1.3 Exemplo

O fragmento de código seguinte é um exemplo de um conteúdo criptografado. O conteúdo criptografado substitui o conteúdo em texto limpo do documento XML. Neste caso, o fragmento representa informações de pagamento. O nome no cartão de crédito (linha 2) e o limite (linha 3) são transmitidos claramente. Entretanto, o número do cartão de crédito é criptografado. O algoritmo de criptografia é o 3DES (linha 5). A chave para descriptografar o número do cartão de crédito é a *MyCompany* (linha 7). Neste caso, a chave para *MyCompany* foi pré-definida com o receptor da mensagem. Os dados criptografados estão contidos no elemento *CipherValue* (linha 10).

Listagem 3.1: Exemplo do *XML Encryption*

```
1  <PaymentInfo xmlns="http://example.org/paymentv2">
2    <Name>John Smith</Name>
3    <CreditCard Limit="5.000" Currency="USdollars">
4      <EncryptedData xmlns="http://www.w3.org/2001/01/xmlenc#" Type="http
         ://www.w3.org/2001/04/xmlenc#Content/">
5        <EncryptionMethod Algoritm="http://www.w3.org/2001/04/xmlenc#3des-
           cbc"/>
6        <ds:KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
7          <ds:KeyName>MyCompany</ds:KeyName>
8        </ds:KeyInfo>
9        <CipherData>
10         <CipherValue>A23B45C56</CipherValue>
11       </CipherData>
12     </EncryptedData>
13   </CreditCard>
14 </PaymentInfo>
```

3.2 Especificação *XML Signature*

A especificação *XML Signature* (ou somente XML-DSig) [10] define como dados digitais são assinados e como a assinatura deve ser representada em XML.

A especificação XML-DSig define o processo para criação e representação de uma assinatura XML e de verificação desta assinatura. Podemos utilizar algoritmos existentes para a assinatura, sumários de mensagens e autenticação de mensagens. Ela oferece várias alternativas para certificados, inclusive X.509. Ela também pode ser usada sem certificados.

A XML-DSig referencia padrões de transformações de dados, como canonização[4], renderização dos dados de acordo com um padrão que elimina diferenças sem importância em representação, e codificação e decodificação.

A especificação XML-DSig não obriga ao uso de algoritmos e de tamanhos de chaves específicos. É de responsabilidade do usuário assegurar-se que as escolhas corretas estão sendo feitas. Os usuários devem considerar cuidadosamente o quanto as assinaturas e os algoritmos utilizados devem ser seguros, e decidir pelo tamanho de chave apropriado.

3.2.1 Formato e Estrutura da XML-DSig

O *XML Signature* basicamente consiste em dois elementos obrigatórios, o elemento *SignedInfo* e o *SignatureValue*. Ainda há mais dois elementos opcionais, o elemento *KeyInfo* e o *Object*.

O elemento *SignedInfo* inclui o *CanonicalizationMethod*, os algoritmos (tipicamente um algoritmo de sumário e um algoritmo de assinatura) usados para produzir a assinatura, e uma ou mais referências para os dados a serem assinados, o algoritmo de transformação dos dados, um identificador para o algoritmo de sumário utilizado com os dados referenciados, e o valor do sumário da mensagem.

O elemento *SignatureValue* representa o valor da assinatura digital e é codificado de acordo com o base64.

O elemento *KeyInfo* fornece as informações necessárias para o receptor validar a assinatura. Se este elemento for omitido, o receptor deve conhecer como validar a assinatura. Por exemplo, duas pessoas podem ter trocado as chaves anteriormente por outros meios, assim, eliminando a necessidade de incluir a chave pública com elemento-filho de *KeyInfo*. Senão, *KeyInfo* poderá conter o identificador da chave, a chave pública

[4]Canonização é o processo de converter dados que possuem mais de uma possível representação em um padrão único de representação.

do assinante, uma referência para a localização da chave pública ou o certificado da chave pública do assinante. Vários formatos de certificados de chaves públicas são suportados.

O elemento *Object* traz qualquer outra informação necessária para o suporte da assinatura.

3.2.2 Transformações de Dados

Assinaturas digitais são dependentes da representação dos dados a serem assinados. Uma alteração sem importância no documento, digamos a adição ou remoção de um espaço em branco, pode ser lido por um sistema como uma alteração significante e pode causar que uma assinatura não possa ser verificada. Para evitar esta possibilidade, documentos XML podem ser transformados em um padrão de representação antes de ser assinado ou verificado.

Cada elemento *reference* da estrutura *SignedInfo* pode conter transformações que são aplicadas para os elementos referenciados. Uma ou mais transformações podem ser especificadas em cada elemento referenciado. A entrada para a primeira transformação é a data identificada pelo atributo URI do elemento *reference*. E essa saída é a entrada da segunda transformação, até que a última transformação gere a entrada para o algoritmo de sumário de mensagem.

A *XML Signature* não obriga o uso de algoritmos de transformação específicos, entretanto, a funcionalidade que eles fornecem é necessária para assegurar que as assinaturas digitais funcionem corretamente. Mesmo que os usuários não queiram usar estes algoritmos em específico, um algoritmo alternativo equivalente deve ser utilizado. Mas, o uso de algoritmos alternativos não é encorajado pois isto limitaria a interoperabilidade do *XML Signature*.

3.2.3 Canonização XML

Canonical XML [1] fornece um método padrão de determinar se dois documentos XML são idênticos. Ele define regras para a transformação de um documento XML em uma representação padrão. Outro documento com a mesma representação canônica será considerado idêntico ao primeiro.

A representação canônica de um documento é uma representação cômoda para

se assinar, já que as regras canônicas aplicadas ao documento XML recebido elimina alterações inconseqüentes. XML Canônico transforma os dados utilizando o algoritmo de codificação UTF-8. A *Canonical XML* normaliza quebras de linhas e atributos, substitui referências, remove referências de *namespaces* desnecessárias, adiciona atributos padrões, e realiza outras funções que eliminam construções desnecessárias e resolve ambigüidades em potencial.

Quando usada com assinaturas digitais, a canonização transforma os dados antes de assiná-los. Então, é usada para transformar os dados novamente antes de verificá-los, assim, elimina a possibilidade de que a verificação possa falhar de forma desnecessária. Já que a canonização pode consumir recursos computacionais, somente porções do documento que serão assinados podem ser canonizadas.

3.2.4 Processo de Assinatura e Verificação Digital na XML-DSig

A especificação XML-DSig aconselha o seguinte procedimento para Assinar um documento:

1. Aplicar os algoritmos de transformação de dados para os objetos de dados a serem assinados. As transformações são aplicadas na ordem que elas foram especificadas;
2. Calcular os sumários de mensagens das saídas das transformações;
3. Criar o elemento de referência que inclui o URI do objeto de dados (opcional), as transformações usadas, o algoritmo de sumário, e o valor do sumário. Algumas referências de elementos podem ser necessárias e criadas. Isto ocorre se uma assinatura encobre vários nós do documento;
4. Criar o elemento *SignedInfo*. Incluir o *SignatureMethod*, o *CanonicalizationMethod*, as referências geradas anteriormente;
5. Aplicar o *CanonicalizationMethod* para o elemento *SignedInfo*;
6. Usar os algoritmos especificados pelo *SignatureMethod* para criar a assinatura. Isto usualmente significa aplicar um algoritmo de sumário de mensagem para o *SignedInfo* canonizado e assinar o sumário resultante;

7. Criar o elemento de *Signature* que contém o *SignedInfo*, o *SignatureValue*, o *KeyInfo* (se necessário), o *Object* (se necessário). Note que um algoritmo de canonização ou de sumário de mensagem pode ser aplicado a diferentes elementos referenciados.

A XMl-DSig aconselha o seguinte procedimento para verificar um documento:

1. Canonizar o elemento *SignedInfo* de acordo com o método especificado no *CanonicalizationMethod*;
2. Obter o objeto de dados referenciado;
3. Processar cada objeto de dados de acordo com as transformações especificadas;
4. Sumarizar o resultado de acordo com o algoritmo de sumário especificado para o elemento referenciado. Comparar o resultado com o valor armazenado no elemento correspondente. Se o dois não são iguais, a verificação falhou;
5. Obter a informação de chave necessária. Isto deve ser disponível na *KeyInfo* ou ela deve ter sido pré-definida;
6. Aplicar o método de assinatura usando a chave obtida para confirmar o *SignatureValue* sobre o elemento *SignedInfo*.

3.2.5 Exemplo

A Listagem 3.2 mostra um exemplo de uma assinatura XML. O elemento *SignatureMethod* (linha 4) especifica o algoritmo de assinatura e o elemento *DigestMethod* (linha 9) especifica o algoritmo de sumário de mensagem. A canonização XML que é usada para transformar a entrada também é informada (linha 3). Os dados a serem assinados são identificados pelo atributo URI de *Reference* (linha 5). Neste caso, existe somente um elemento *Reference*, e ele identifica um documento separado chamado *order*. O elemento *DigestValue* (linha 10) contém o sumário da mensagem. O elemento *SignatureValue* (linha 12) é calculado baseada nos elementos filhos de *SignedInfo* (linha 2 a 13). Finalmente, a chave pública, que é usada para verificar a assinatura é anexada (linha 14 a 19).

Listagem 3.2: Exemplo do *XML Signature*

```
1 <Signature xmlns="http://www.w3.org/200/09/xmldsig#">
2   <SignedInfo>
```

```
3     <CanonicalizationMethod Algoritm="http://www.w3.org/TR/2001/REC-xml-
          c14n-20010315"/>
4     <SignatureMethod Algoritm="http://www.w3.org/2000/xmldsig#ds-sha1"/>
5     <Reference URI="http://www.mycompany.com/order/"/>
6     <Transforms>
7       <Transform Algoritm="http://www.w3.org/TR/2001/REC-xml-c14n
            -20010315"/>
8     </Transforms>
9     <DigestMethod Algoritm="http://www.w3.org/2000/09/xmldsig#sha1"/>
10    <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
11  </SignedInfo>
12  <SignatureValue>MC0CFFrVLtRlK ...</SignatureValue>
13  <KeyInfo>
14  <KeyValue>
15    <DSAKeyValue>
16      <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
17    </DSAKeyVAlue>
18    </KeyValue>
19  </KeyInfo>
20 </Signature>
```

3.3 Especificação XKMS

A especificação XKMS (*XML Key Management Specification*) [14] define protocolos para que aplicações baseadas em XML possam incorporar os serviços de confiança de um esquema PKI (*Public Key Infrastructure*). A especificação XKMS foi publicada pela *VeriSign* [39], *Microsoft* [24] e *webMethods* [41]; e submetida para a W3C [40]. A especificação XKMS possui a vantagem de encapsular toda a complexidade do esquema PKI habilitando as aplicações clientes a acessarem serviços responsáveis pelo processamento das funcionalidades do esquema PKI.

A especificação XKMS fornece duas categorias de funcionalidades de esquema PKI:

- Gerenciamento de ciclo de vida das chaves públicas, que compreende as funcionalidades de registro, revogação e renovação de uma chave pública;
- Validação e localização de uma chave pública.

Os protocolos de gerenciamento de ciclo de vida das chaves públicas são definidos na especificação chamada X-KISS e os protocolos de validação e localização são definidos na especificação chamada X-KRSS. Assim, a especificação XKMS é formada por duas outras especificações, a X-KISS e a X-KRSS[5].

3.3.1 Especificação X-KISS

A especificação X-KISS (*XML Key Information Service Specification*) define um protocolo para o serviço de confiança que resolve as informações de chave pública contidas nos elementos da especificação *XML-DSig*. Os protocolos da *X-KISS* habilitam os clientes a delegarem as tarefas necessárias para processar os elementos *ds:KeyInfo*. A especificação *X-KISS* minimiza a complexidade das aplicações permitindo que estas se tornem clientes da própria especificação e serem poupadas da complexidade dos serviços de confiança do PKI.

Um cliente recebe um documento XML assinado. O elemento *KeyInfo* na assinatura especifica um método de recuperação para um certificado X.509. O cliente, que não possui meios para acessar o certificado X.509 via URL ou para analisá-lo de modo a obter os parâmetros da chave pública, delega essas tarefas ao serviço de confiança.

O serviço de validação permite que um cliente delegue todas as funções de processamento de confiança a um serviço de confiança. Como ocorre com o serviço de localização, o cliente cria uma requisição que especifica a informação que o serviço de validação deve localizar. No entanto, ao contrário do serviço de localização, o de validação é responsável por garantir a credibilidade dos dados retornados antes de confiar neles.

Um cliente recebe um documento XML assinado e requisita que o serviço de confiança determine se a chave de assinatura é confiável. Nesse caso, um certificado X.509 autentica a chave de assinatura. O serviço de confiança constrói um caminho de certificados confiáveis e então valida cada um dos certificados encontrados por consulta à Lista de Certificados Revogados relevante. O cliente fica protegido da complexidade e o serviço de confiança retorna apenas as informações de interesse específico para o cliente: os parâmetros de chave, os dados vinculados à chave e a validade dessa vinculação.

A delegação das funções de processamento de confiança a um serviço específico

[5]A especificação X-KISS e a X-KRSS não correspondem a especificações propriamente ditas, mas são "subespecificações" da XKMS.

de confiança torna possível o controle através da empresa e a supervisão da configuração da PKI. Isso é essencial para as aplicações B2B (*Business-to-Business*), onde os relacionamentos de confiança importantes ocorrem entre as empresas e não entre os indivíduos ou entre as aplicações por eles utilizadas.

3.3.2 Especificação X-KRSS

A especificação X-KRSS (*XML Key Registration Service Specification*) define um protocolo para o serviço de confiança que aceita o registro de informações sobre as chaves públicas. Uma vez registrada, a chave pública pode ser usada em conjunto com outros serviços *web*, incluindo a especificação X-KISS.

A X-KRSS foi projetada para suportar todas as funções associadas com o ciclo de vida da chave pública:

- Registro: a função registro suporta o registro da associação entre uma chave pública e dados adicionais (como um nome) para criar uma "vinculação de chave". As chaves privadas podem ser geradas localmente pelo cliente (desejável para chaves de assinatura) ou por um serviço central de geração de chaves (desejável em casos onde a recuperação de chaves é suportada). As requisições podem ser autenticadas através de um segredo compartilhado de uso limitado ou por uma assinatura digital;

- Renovação: a especificação XKMS permite que uma PKI seja operada sem que sejam emitidos certificados digitais, eliminando a necessidade de renová-los. Em casos onde os certificados são emitidos pela PKI subjacente, o processamento das renovações pode ser executado automaticamente, sem necessidade de interação com o cliente;

- Revogação: uma parte autorizada pode requerer que o serviço de confiança revogue uma vinculação de chave. Isso pode ser necessário porque a chave foi comprometida, ou porque as informações contidas na vinculação de chave são incorretas;

- Recuperação: a recuperação de chaves privadas é essencial quando um usuário final extravia sua chave privada e precisa ter acesso aos seus dados criptografados. A função de recuperação da X-KRSS oferece um meio autenticado para re-emitir a chave privada para um usuário.

A especificação X-KRSS pode ser configurada hierarquicamente, do mesmo modo que uma Autoridade de Registro Local. Isso permite que um pedido de registro seja autenticado por um serviço local de confiança e então repassado para outro serviço de confiança, onde o processamento é executado.

3.4 Especificação *WS-ReliableMessage*

O foco do *WS-ReliableMessage*[6] [2], [11] é relativamente estreito, quando comparada a outras especificações. Sua principal preocupação é a garantia de entrega de mensagens SOAP. Além da garantia da entrega de uma mensagem, também pode haver a garantia que as mensagens foram entregues na ordem que elas foram enviadas.

A especificação *WS-ReliableMessage* (ou somente WS-RM) estabelece um método padrão para a notificação do remetente de uma mensagem sobre o sucesso ou a falha na entrega da mensagem. Esta notificação é feita através de uma confirmação (Figura 3.1) que é enviada através de uma mensagem separada ou com uma construção anexada à resposta da mensagem gerada pelo receptor da mensagem.

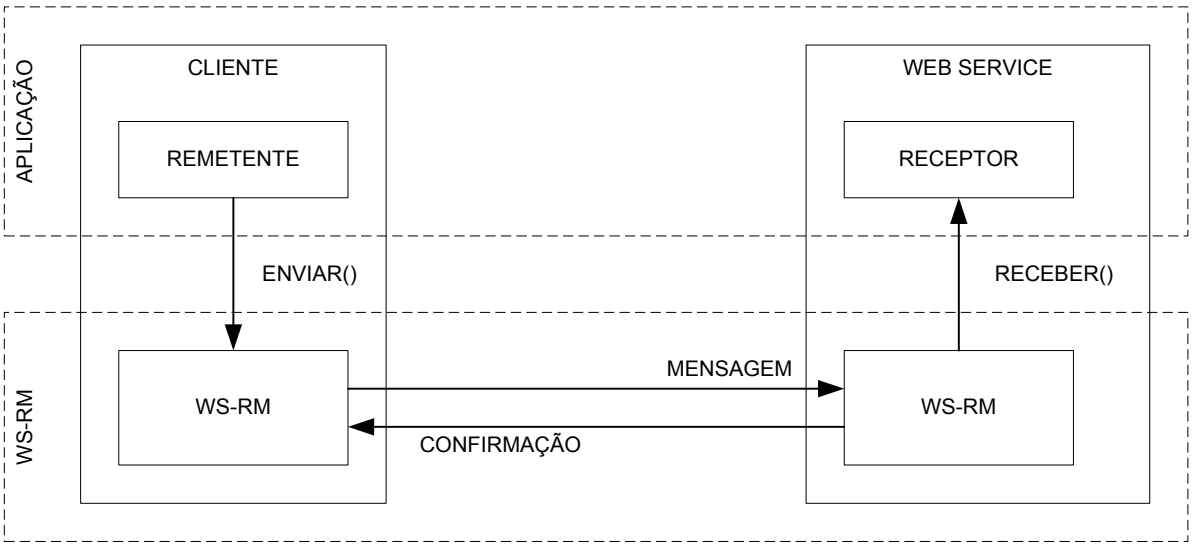


Figura 3.1: Esquema do WS-RM

[6]Não confundir *WS-ReliableMessaging* com seu concorrente, a especificação *WS-Reliability*.

3.4.1 Sequências

Para implementar um sistema de entrega confiável de mensagens, a especificação WS-RM requer que os cabeçalhos SOAP tenham elementos específicos com um número identificador de sequência atribuído às mensagens. A informação de uma sequência é usada para identificar e processar a mensagem.

3.4.2 Políticas de Garantia de Entrega

Existem 4 tipos de políticas de garantia de entrega que podem ser utilizadas junto com a WS-RM:

1. No Máximo Uma Vez (*AtMostOnce*): garante que uma mensagem só pode ser entregue uma vez ou nenhuma. A Figura 3.2 (a) representa um cenário desta política, onde uma entrega não realizada não é seguida por outra tentativa de envio;

2. Pelo Menos Uma Vez (*AtLeastOnce*): uma mensagem pode ser entregue mais de uma vez, mas assegura que foi entregue pelo menos uma vez. Nesta política, uma mensagem pode ser entregue duas vezes (Figura 3.2 (b));

3. Exatamente Uma Vez (*ExactlyOnce*): assegura que uma mensagem foi entregue uma única vez. Nesta política, uma tentativa de retransmissão ocorre, se houver uma falha de entrega (Figura 3.2 (c));

4. Em Ordem (*InOrder*): pode suplementar qualquer um dos anteriores e assegura também que as mensagens foram entregues na sequência em que elas foram mandadas. Na Figura 3.2 (d), a entrega inicial da mensagem 2 falhou e foi reenviada, como a seqüência deve ser completada.

3.4.3 Confirmações

Para o receptor de uma mensagem notificar o remetente que as mensagens de uma seqüência foram entregues com sucesso, ele responde usando mensagens de confirmações (Figura 3.3 (a)).

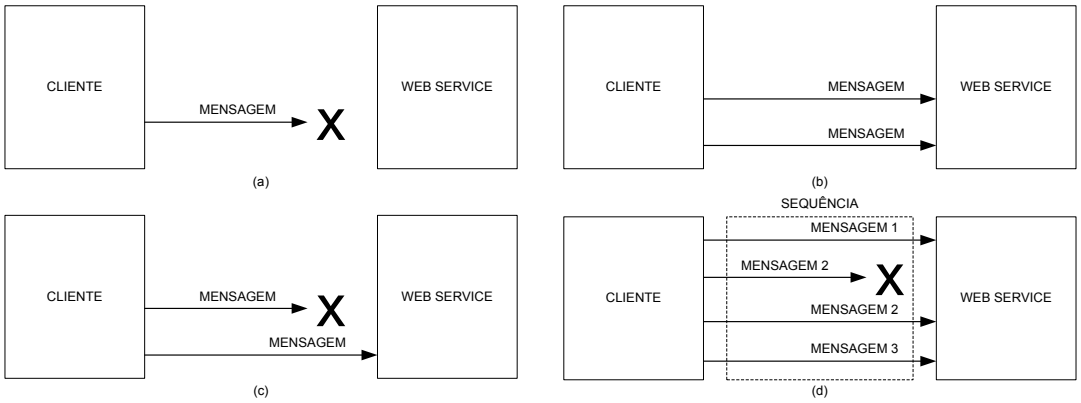


Figura 3.2: Políticas do WS-RM

Uma confirmação pode ser enviada por cada mensagem recebida. Alternativamente, pode-se escolher enviar uma confirmação somente ao término da entrega total de uma seqüência (Figura 3.3 (b).

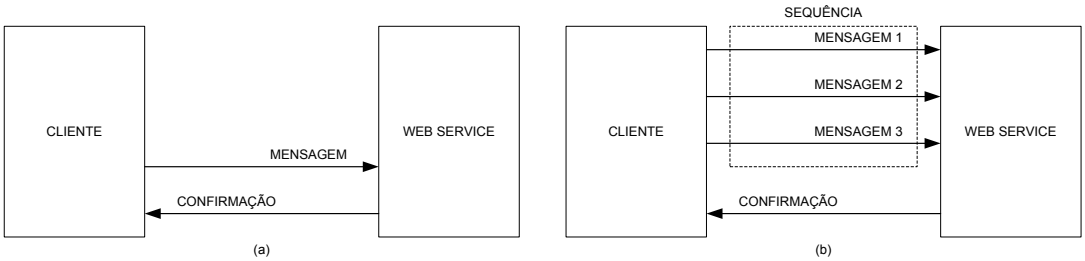


Figura 3.3: Confirmações do WS-RM

3.4.4 Formato e Estrutura da WS-RM

A especificação WS-RM cria elementos nos cabeçalhos das mensagens SOAP para que possam ser processados pelos receptores.

O elemento *Sequence* agrupa informações de troca de mensagens confiável. Na Listagem 3.3, o elemento *Sequence* usa o elemento *Identifier* para associar uma URI única para a seqüência. O elemento *MessageNumber* determina a posição da mensagem corrente de acordo com a ordem interna da seqüência. Este número incrementa com cada mensagem subseqüente enviada como parte da sequência. Finalmente, *LastMessage* identifica que a mensagem é a última a ser entregue.

Listagem 3.3: Elemento *Sequence* do WS-RM

```
1 <wsrm:Sequence>
2   <wsu:Identifier>http://www.examples.ws/</wsu:Identifier>
```

```
3   <wsrm:MessageNumber>2</wsrm:MessageNumber>
4   <wsrm:LastMessage/>
5 </wsrm:Sequence>
```

O elemento *SequenceAcknowledgement* pode ser enviado em uma mensagem separada, ou como parte de uma mensagem de resposta. O elemento também abriga os elementos *Identifier* e *SequenceAcknowledgementRange*. O elemento *SequenceAcknowledgement* informa ao remetente qual das mensagens na seqüência foram entregues com sucesso, especificando uma variação que corresponde ao número de mensagens na seqüência. A Listagem 3.4 representa o uso do elemento *SequenceAcknowledgement*.

Listagem 3.4: Elemento *SequenceAcknowledgement* do WS-RM

```
1 <wsrm:SequenceAcknowledgement>
2   <wsu:Identifier>http://www.examples.ws/</wsu:Identifier>
3   <wsrm:AcknowledgementRange Upper="6" Lower="1">
4 </wsrm:SequenceAcknowledgement>
```

Na Listagem 3.4, todas as 6 mensagens foram entregues com sucesso. Se uma das mensagens na seqüência falhar em sua entrega, o elemento *SequenceAcknowledgement* precisa abrigar uma quantidade de *AcknowledgementRange* para identificar somente aquelas que foram entregues. Ausências na seqüência são consideradas falhas de entregas.

Listagem 3.5: Confirmações de Entregas no WS-RM

```
1 <wsrm:SequenceAcknowledgement>
2   <wsu:Identifier>http://www.examples.ws/</wsu:Identifier>
3   <wsrm:AcknowledgementRange Upper="3" Lower="1">
4   <wsrm:AcknowledgementRange Upper="6" Lower="5">
5 </wsrm:SequenceAcknowledgement>
```

Na Listagem 3.5, as mensagens 1, 2, 3, 5 e 6 foram entregues com sucesso. Por omissão, a transmissão da mensagem 4 é considerada como uma falha.

3.5 Padrão RDF

O RDF (*Resource Description Framework*) [5], [13], [21], [22] corresponde a uma linguagem utilizada para representar as informações de recursos localizados na

Internet. O padrão RDF é destinado a representar metadados de recursos da *web* como título, autor e data de modificação de uma página e direitos de cópia. Entretanto, como podemos generalizar o conceito de um *"recurso da web"*, o RDF pode também ser utilizado para representação de informações de coisas que podem ser identificadas na Internet, mesmo quando estas coisas não podem ser diretamente obtidas pela *web*. Podemos tomar como exemplo, informações de preços e especificações de produtos de lojas virtuais; ou as informações das preferências de entrega de algum cliente destas lojas.

O padrão RDF é destinado para situações em que estas informações necessitam ser processadas por aplicações ao invés de serem somente mostradas para o usuário. O RDF fornece um *framework* comum para expressar estas informações e então poderem ser trocadas entre aplicações sem perda de significado. Desde que estamos falando de um *framework* comum, os desenvolvedores de aplicações podem aproveitar interpretadores RDF existentes. A habilidade de trocar informações entre diferentes aplicações significa que as informações podem ser disponibilizadas para qualquer outra aplicação que possua o suporte ao RDF.

O padrão RDF é baseado na idéia de identificar coisas usando URIs (*Uniform Resource Identifiers*) e de descrever recursos em termos de simples propriedades e valores. Esta característica habilita o RDF a representar simples sentenças de recursos como gráficos de nós e arcos representando os recursos e suas propriedades e valores. As informações expressas por URI podem representar:

- Informações individuais, por exemplo, Bob;
- Tipos de coisas, por exemplo, pessoa;
- Propriedades das coisas, por exemplo, *e-mail*;
- Valores das propriedades, por exemplo, "mailto:bob@ufma.br", como valor da propriedade *e-mail* (O RDF pode usar *strings* como "Bob", e outros tipos de dados como inteiros e datas, como valores de uma propriedade).

O RDF também fornece uma sintaxe baseada em XML (RDF/XML) para a expressão das sentenças. Utilizando os exemplos, podemos criar uma sentença "existe uma Pessoa identificada por `http://www.w3.org/People/EM/contact#me` cujo nome é 'Bob', cujo *e-mail* é 'bob@ufma.br', e cujo título é 'Dr'.". A representação em XML desta representação está ilustrada na Listagem 3.6.

Listagem 3.6: Representação de uma sentença RDF em XML

```
1 <?xml version="1.0"?>
2 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:
    contact="http://www.w3.org/2000/10/swap/pim/contact#">
3   <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
4     <contact:fullName>Bob</contact:fullName>
5     <contact:mailbox rdf:resource="mailto:bob@ufma.br"/>
6     <contact:personalTitle>Dr.</contact:personalTitle>
7   </contact:Person>
8 </rdf:RDF>
```

3.6 JADE-S: Um *Add-on* de Segurança para JADE

O JADE-S (*JADE Security*) corresponde a um *add-on* disponível para o JADE que fornece mecanismos de segurança como criptografia, assinatura digital, autenticação e autorização.

Os agentes de uma plataforma JADE que utilizam o JADE-S possuem uma chave pública e privada para os procedimentos de criptografia e assinatura digital das mensagens.

O mecanismo de autenticação é baseado no JAAS (*Java Authenticaion and Authorization Service*) [34] que fornece mecanismos de autenticação e de autorização para aplicações Java.

A criptografia e a assinatura digital são aplicadas ao *payload* da mensagem. As informações referentes a segurança (como os algoritmos de criptografia e assinatura digital utilizados) são colocadas no envelope da mensagem. A estrutura de uma mensagem está ilustrado na Figura 3.4.

O JADE-S possui a limitação de somente funcionar em uma mesma plataforma de agentes.

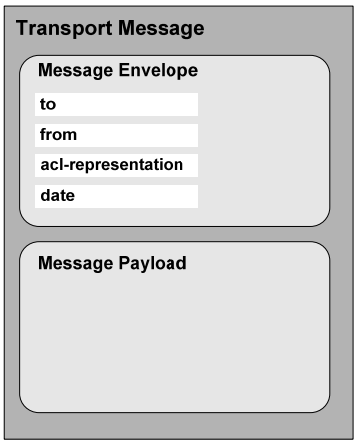


Figura 3.4: Estrutura de uma mensagem

3.7 JMS-MTP: Garantia de Entrega de Mensagens para JADE

Basicamente, o JMS-MTP [3] trata-se de um MTP para o JADE baseado no JMS (*Java Message Service*) [35]. O *Java Message Service* é um padrão de comunicação que permite aplicações J2EE criar, receber e ler mensagens. O JMS também fornece padrões para troca de mensagens confiável para aplicações corporativas. O JMS-MTP utiliza uma API do JMS implementada em Java.

A principal vantagem do JMS-MTP sobre os atuais MTPs é a garantia de entrega de mensagens. O JMS-MTP fornece suporte para novas tentativas de entrega de mensagens que falharam e filas de mensagens para quando uma plataforma esteja fora de contato.

O JMS-MTP resumidamente funciona da seguinte forma:

1. Uma nova plataforma JADE inicia o JMS-MTP;
2. O JMS-MTP ao iniciar, conecta-se a um servidor JMS;
3. Se não existir uma fila corresponde a plataforma que iniciou o JMS-MTP, então é criada uma nova fila para aquela plataforma;
4. Quando uma mensagem é enviada a outra plataforma, a mensagem é encaminhada à fila corresponde à plataforma de destino no servidor JMS;
5. O JMS-MTP retira as mensagens da fila correspondente a sua plataforma para que

as mensagens possam ser entregues aos agentes.

3.8 Conclusões

Neste capítulo, apresentamos as especificações XML que são adaptadas nas soluções de segurança e de confiabilidade propostas neste trabalho.

Nós apresentamos a especificação XML-Enc (*XML Encryption*) e a XML-DSig (*XML Signature*) para, respectivamente, criptografia e assinatura digital de documentos XML. Também apresentamos a especificação XKMS que fornece suporte ao esquema PKI aos *Web Services*.

Nós também apresentamos o padrão RDF que fornece uma representação XML do conteúdo das mensagens trocadas entre os agentes em JADE.

Nós também discutimos as soluções de segurança e de confiabilidade existentes para o *framework* JADE. Nós apresentamos o JADE-S, que consiste em um *add-on* para fornecer mecanismos de criptografia e assinatura digital para o JADE, e o JMS-MTP, que fornece garantia de entrega de mensagem para agentes em JADE.

4 Modelagem e Implementação da Solução de Segurança

Este capítulo tem por objetivo apresentar em um nível mais detalhado a solução de segurança proposta nesta dissertação. A visão geral da solução e suas particularidades são apresentadas neste capítulo, bem como, o esquema PKI, a criptografia e a assinatura digital suportados pela solução. Neste capítulo, nós apresentamos também uma proposta de um modelo de segurança específico ao servidor XKMS. Nós apresentamos a modelagem e prototipagem da solução de segurança e ilustramos a utilização da solução com um exemplo. Um comparativo entre nossa proposta de segurança e o JADE-S também é apresentado. Finalmente, nós apresentamos as conclusões a cerca da solução de segurança proposta, destacando os benefícios e limitações.

4.1 Visão Geral da Solução de Segurança

A solução de segurança proposta neste trabalho [28] possui os seguintes mecanismos de segurança: autenticação, confidenciabilidade, não-repúdio do remetente e integridade. O modelo proposto adapta várias especificações XML de segurança para fornecer estes mecanismos. A confidenciabilidade é obtida através da criptografia realizada pela especificação XML-Enc [9]. A integridade, autenticação e o não-repúdio do remetente são obtidos através das assinaturas digitais fornecidas pela especificação XML-DSig [10]. Tanto a especificação XML-Enc quanto a especificação XML-DSig utilizam chaves públicas e privadas que serão gerenciadas pela especificação XKMS [14]. Nós propomos, neste trabalho, um esquema de segurança específico para o servidor XKMS baseado em chaves secretas geradas a partir do algoritmo do *Diffie-Hellman*.

4.2 Esquema PKI Suportado

Como uma das especificações adaptadas nesta solução é a especificação XKMS, o modelo é similar ao esquema PKI (*Public Key Infrastructure*) [8], [42]. Assim, os agentes

iniciam suas atividades e geram um par de chaves. Um agente gera somente um par de chaves (pelo algoritmo RSA) que consiste em uma chave pública e uma chave privada.

O papel da chave pública é ser distribuída para outros agentes que devem participar da comunicação segura. A chave privada deve ser mantida em segredo pelo seu proprietário.

Como os agentes necessitam conhecer a chave pública do destinatário (ou receptor) para transmitir uma mensagem segura, um servidor XKMS deve ser disponibilizado para todos os agentes.

O papel do servidor XKMS é receber várias requisições de chaves e retornar as respostas apropriadas. Após um agente gerar seu par de chaves, ele pode registrar sua chave pública junto ao servidor XKMS para habilitar outros agentes a localizarem e a utilizarem sua chave pública registrada.

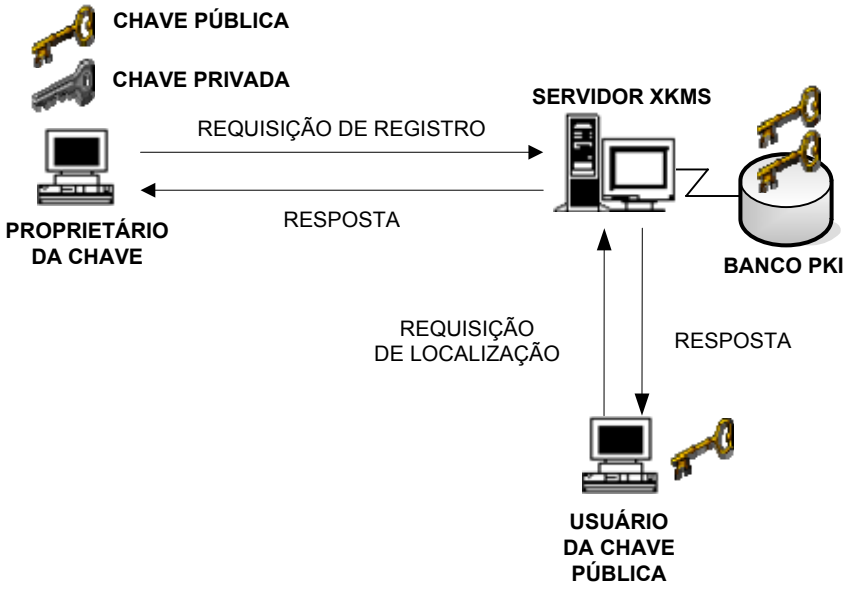


Figura 4.1: Esquema PKI suportado pela solução de segurança

Na Figura 4.1, um agente, para registrar sua chave pública, faz uma requisição de registro de chave e manda a chave pública para um servidor XKMS. O servidor usa a requisição e armazena a chave pública e os dados da identificação do proprietário desta chave.

Qualquer agente pode localizar uma chave pública de outro agente em um servidor XKMS. Para localizar uma chave pública, um agente faz uma requisição de localização com a identificação do proprietário da chave pública desejada e a envia para o servidor XKMS. O servidor XKMS consulta a base de dados PKI em busca da chave

pública correspondente à identificação do proprietário. Ao encontrar a chave pública, o servidor XKMS a retorna para o agente requisitante (Figure 4.1).

4.3 Criptografia Suportada

Conforme a Figura 4.2, o par de chaves é usado no processo de criptografia e descriptografia das mensagens. Uma mensagem criptografada por uma chave pública só pode ser descriptografada usando a chave privada correspondente. Um agente que necessita transmitir uma mensagem criptografada deve conhecer a chave pública do destinatário. Se o agente não conhecer a chave pública do destinatário da mensagem, ele pode localizar a chave pública no servidor XKMS. Todos os agentes devem registrar suas chaves públicas no servidor XKMS a fim de participarem de uma comunicação segura. Um agente que receber uma mensagem criptografada pode descriptografá-la utilizando sua própria chave privada.

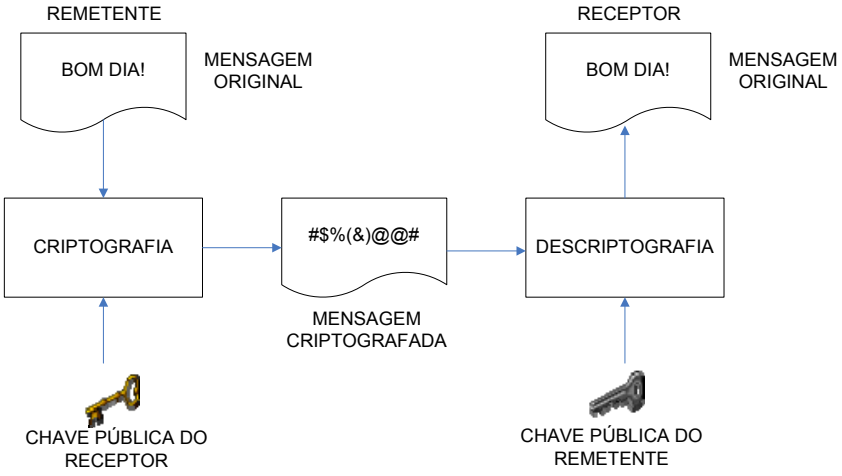


Figura 4.2: Criptografia suportada pela solução

Criptografia por chave pública é mais segura que a criptografia de chave secreta, mas os algoritmos de chave pública são muito mais custosos do que os algoritmos de criptografia de chave secreta. Para resolver este problema, utilizamos um processo de criptografia e descriptografia otimizado.

A Figura 4.3 mostra o processo de criptografia e descriptografia otimizado, em que ao invés do conteúdo da mensagem ser criptografado com a chave pública, nós usamos outra chave (uma chave assimétrica, também chamada de chave secreta) para criptografar o conteúdo da mensagem. Esta chave precisa ser conhecida pelo destinatário, então deve

ser enviada criptografada junto com a mensagem. A chave pública do destinatário é utilizada para criptografar a chave secreta. A mensagem e a chave, ambas criptografadas, são enviadas para o destinatário, que utiliza sua chave privada para descriptografar a chave recebida e com ela descriptografar a mensagem. Assim, o custoso algoritmo de chave pública não é usado para criptografar grandes porções de dados, ao invés disso, o algoritmo é usado para criptografar a outra chave usada para criptografar a mensagem.

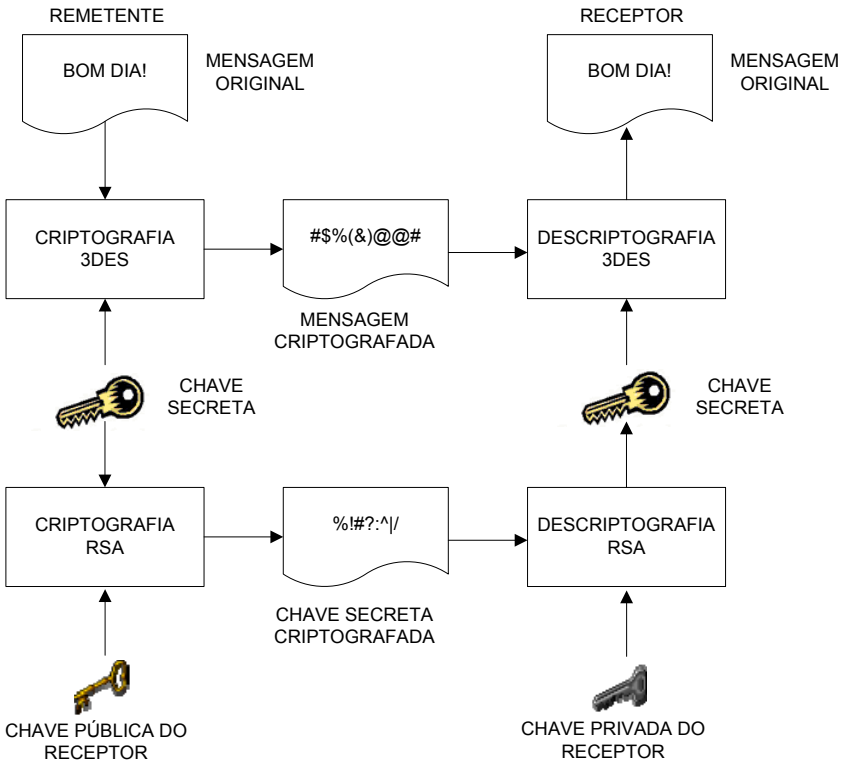


Figura 4.3: Criptografia otimizada com uma chave secreta

4.4 Assinatura Digital Suportada

Na solução proposta, a segurança é intensificada pelo modelo de assinatura digital. Um agente assina digitalmente a mensagem criptografada, assim, o destinatário pode verificar se a mensagem não foi alterada durante a transmissão ou se a mensagem foi realmente enviada pelo remetente.

Para assinar ou verificar uma mensagem, um agente usa o mesmo par de chaves gerado que foi usado para o processo de criptografia. Já que as chaves privadas são dados secretos que só podem ser acessados pelo próprio agente, elas são usadas no processo de assinatura digital.

No processo de assinatura digital, o remetente gera um código de *hash* usando sua chave privada e a anexa junto à mensagem.

O *hash* em uma mensagem assinada só pode ser validado usando a chave pública correspondente à chave privada usada no processo de assinatura digital. Assim, a mensagem pode ser criptografada por um agente usando a chave pública do destinatário e assinada digitalmente usando a chave privada do próprio remetente. A mensagem recebida pode ser verificada pelo receptor usando a chave pública do remetente e descriptografada usando sua própria chave privada.

A Figura 4.4 apresenta a assinatura digital suportada pela solução junto com a criptografia suportada.

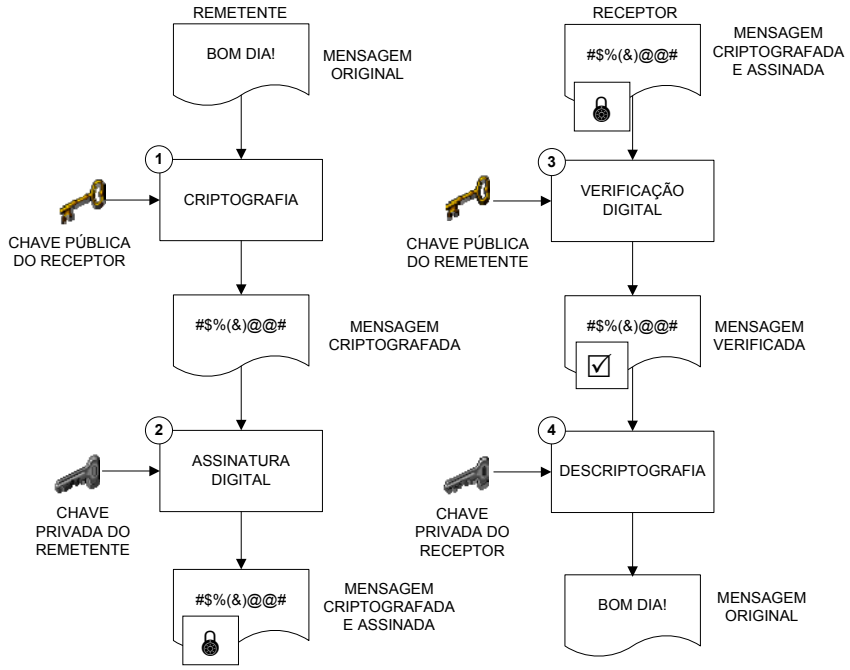


Figura 4.4: Assinatura digital e criptografia suportadas pela solução de segurança

4.5 Segurança do Servidor XKMS

Um intruso poderia registrar sua própria chave pública e ainda se passar por um agente e assim conseguir capturar e ler certa parcela da comunicação do sistema. Um intruso poderia também requisitar uma chave pública através do servidor XKMS e enviar uma mensagem para o agente a que pertence a chave pública, tentando confundi-lo. Assim, nosso trabalho também propõe medidas de segurança específicas ao servidor XKMS. A Figura 4.5 ilustra o modelo de segurança do servidor XKMS proposto neste

trabalho. Nesta figura, um agente chamado “agente x” registra sua chave pública no servidor XKMS e logo em seguida realiza uma localização de outra chave pública.

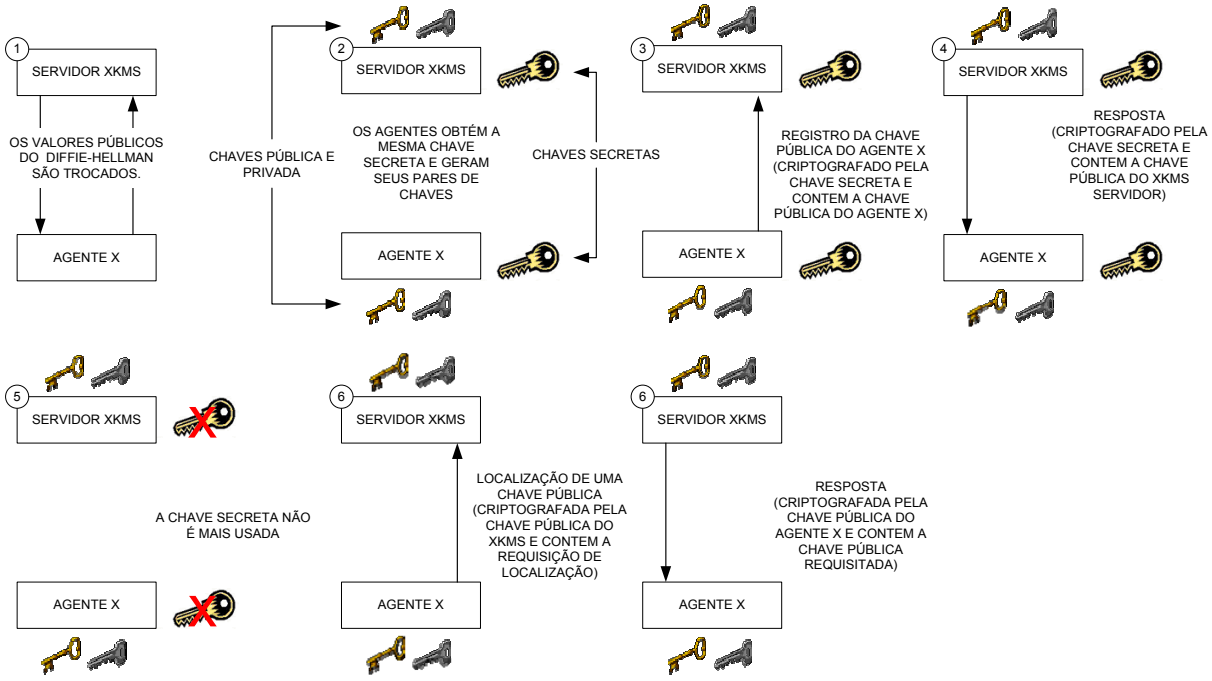


Figura 4.5: Modelo de segurança do servidor XKMS

Na Figura 4.5, O “agente x”, ao ser iniciado, entra em acordo com o servidor XKMS e ambos geram uma mesma chave secreta através do algoritmo *Diffie-Hellman* (passo 1 da Figura 4.5). Sempre que um novo agente é iniciado, uma nova chave secreta para a comunicação entre este agente e o servidor XKMS é gerada. Após isso, o agente pode iniciar suas atividades e gerar separadamente seu par de chaves, pública e privada (passo 2).

A chave secreta é utilizada para criptografar a mensagem de registro da chave pública do “agente x” (passo 3). Como, o servidor XKMS possui a mesma chave secreta, ele pode (e somente ele) descriptografar a mensagem e armazenar a chave pública do agente.

O servidor XKMS responde ao agente informando que a chave pública foi registrada com sucesso e anexa a mensagem de resposta a sua própria chave pública. A mensagem de resposta é criptografada utilizando a chave secreta, assim, somente o “agente x”, que participou da sessão do *diffie-hellman* pode descriptografá-la (passo 4).

Após receber a resposta do registro e a chave pública do servidor XKMS, o “agente x” e o próprio servidor XKMS podem descartar a chave secreta utilizada (passo

5). Todas as mensagens seguintes trocadas entre os agentes e o servidor XKMS serão protegidas utilizando as chaves públicas e privadas dos mesmos.

Para realizar a localização de outra chave pública, o "agente x" criptografa a requisição utilizando a chave publica do servidor XKMS (passo 6) e recebe sua resposta criptografada pela sua própria chave pública (passo 7). Este esquema de segurança impossibilita que um intruso possa registrar uma chave pública falsa ou que possa utilizar uma chave pública válida para qualquer tipo de ataque.

Um ataque do Homem-do-Meio poderia ser utilizado neste modelo de segurança e conseguir iludir tanto o servidor XKMS quanto qualquer agente que esteja registrando ou localizando uma chave pública. Assim, nosso modelo de segurança proposto ao servidor XKMS conta ainda com um arquivo de configuração que é utilizado para evitar o ataque do Homem-do-Meio.

O arquivo de configuração é ilustrado na Listagem 4.1. O arquivo utiliza o padrão XML e contém informações sobre o servidor XKMS e sobre os parâmetros do algoritmo do *Diffie-Hellman* que são lidas pelos agentes e pelo servidor XKMS. As linhas 4 e 5 correspondem respectivamente ao endereço IP e ao endereço MAC do servidor XKMS. Um agente, antes de participar da sessão do *Diffie-Hellman*, registra uma entrada MAC em seu computador através destas configurações utilizando o comando de linha correspondente à plataforma da sua máquina (linha 12). As linhas 8, 9 e 10 correspondem ao parâmetros do *Diffie-Hellman*. Já que os parâmetros não são trocados através da rede, isto representa uma dificuldade a mais para possíveis intrusos. O fato dos agentes possuirem entradas estáticas para o endereço MAC do servidor XKMS impossibilita os ataques do Homem-do-Meio.

Listagem 4.1: Arquivo de configuração do protótipo da solução de segurança

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <configuration>
3    <xkms_server>
4      <ip_address>192.168.4.124</ip_address>
5      <mac_address>00-40-f4-a2-b9-6d</mac_address>
6    </xkms_server>
7    <diffie-hellman>
8      <prime_modulus_P>108953995028355359575[...]</prime_modulus_P>
9      <base_generator_G>98477661151217114125[...]</base_generator_G>
10     <bit_size_exponent_L>1024</bit_size_exponent_L>
```

```
11    </diffie-hellman>
12    <platform>win</platform>
13  </configuration>
```

4.6 Modelagem e Prototipagem

A solução de segurança proposta neste trabalho teve seu protótipo modelado e projetado utilizando a Metodologia de Análise e Projeto Orientado a Objetos e foi implementado como uma API utilizando a linguagem Java.

Nós utilizamos uma *toolkit* chamada TSIK (*Trust Service Integration Kit*) disponibilizada pelo *VeriSign* [39] para a implementação do protótipo. VeriSign é uma companhia que fornece soluções para autenticação e certificação para aplicações coorporativas na Internet. O TSIK atualmente se encontra na versão 1.10 e fornece bibliotecas Java com implementações das especificações XKMS, *XML Encryption* e *XML Signature*. O benefício de usar o TSIK é a implementação rápida dos mecanismos de segurança da solução proposta.

Nossa API utiliza estas bibliotecas que implementam as especificações XKMS, XML-Enc e XML-DSig para fornecer os mecanismos de segurança da solução proposta. A implementação do XKMS fornece os métodos de geração de chaves e do gerenciamento de chave pública, como registro de chave e serviço de localização de chave. A implementação do *XML Encryption* fornece os métodos de criptografia e descriptografia. A implementação do *XML Signature* fornece os métodos para assinatura e verificação digital.

O modelo proposto também necessita que os agentes troquem as mensagens usando o padrão RDF. Existe um *add-on*[1] que habilita os agentes a codificar o conteúdo das mensagens de acordo com a sintaxe baseada em RDF. Este *add-on* fornece o suporte ao RDF necessário ao nosso protótipo.

[1] O *add-on* está disponível no *site* do próprio JADE sob a página *community & developers*. O nome do *add-on* é *RDF-Codec NEW* e a data de liberação é de dezembro de 2003.

4.6.1 Pacotes do Protótipo

O diagrama de pacotes do protótipo de segurança está ilustrado na Figura 4.6. As classes do protótipo da solução de segurança foram agrupadas no pacote *prototype.security* (Figura 4.6). O pacote *prototype.security* possui uma dependência do pacote *jade.lang.acl* por enviar mensagens que representam as requisições de registro e de localização de chaves públicas. O pacote *prototype.security* também precisa manipular ontologias[2] e *codecs* de linguagens de conteúdo[3], então, possui dependências para o pacote *jade.content*. O pacote necessita ainda manipular as representações dos agentes JADE, assim, possui dependências para o pacote *jade.core*. A dependência do pacote *verisign.com* se dá pela utilização da biblioteca de implementações das especificações XML.

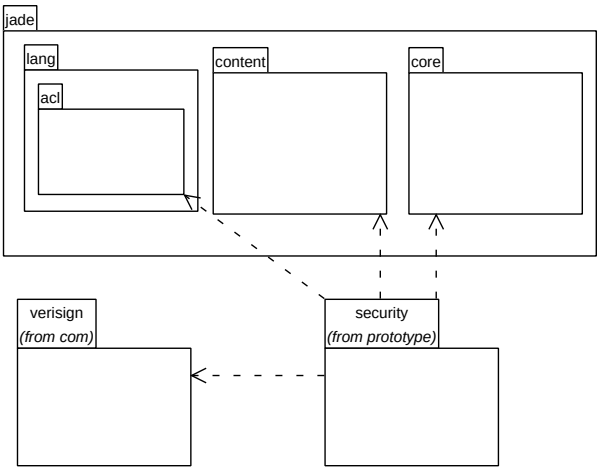


Figura 4.6: Diagrama de pacotes do protótipo da solução de segurança

4.6.2 Classes do Protótipo

O diagrama de classes do protótipo de segurança está ilustrado na Figura 4.7.

O pacote *prototype.security* possui outros pacotes chamados *databasePKI* e *xkmsServer*. O pacote *prototype.security.databasePKI* possui classes que representam uma base de dados PKI. O pacote *prototype.security.xkmsServer* possui classes que simulam um servidor XKMS. O nosso protótipo conta com um agente JADE que fará o papel

[2]Um ontologia corresponde ao vocabulário e semântica que representam o conteudo das mensagens trocadas entre os agentes.

[3]Um *codec* de linguagem de conteúdo trata-se uma biblioteca que fornece métodos para o entendimento de uma linguaem de conteúdo. Ele serve para dois agentes "conversarem" utilizando uma mesma linguagem de conteúdo.

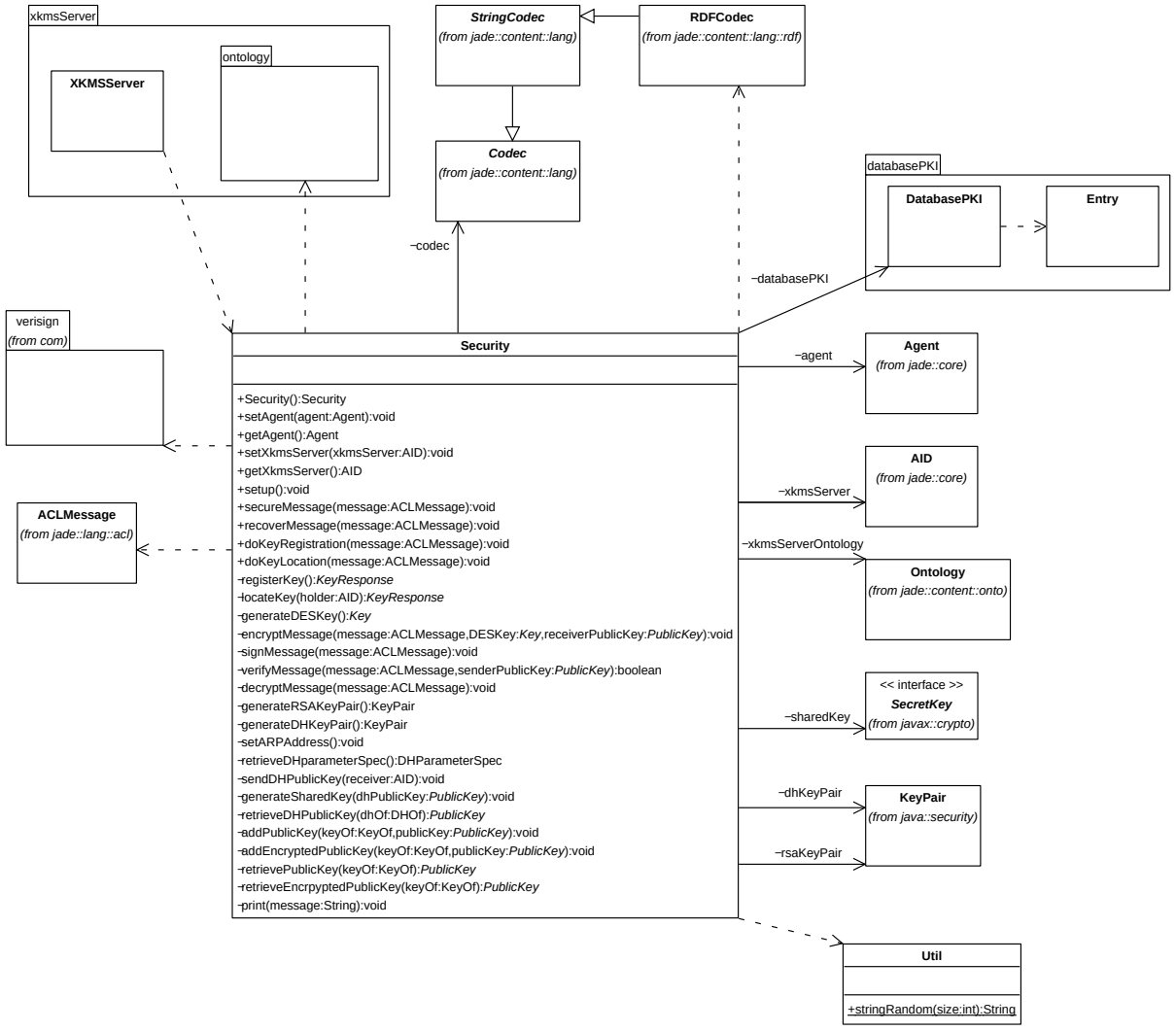


Figura 4.7: Diagrama de classes do protótipo da solução de segurança

de um servidor XKMS responsável por responder as requisições de registro e localização de chave pública de acordo com o comportamento de um servidor XKMS[4]. O pacote *prototype.security.xkmsServer* possui ainda um pacote chamado *ontology* responsável pela definição da ontologia utilizada para as mensagens ACL que representam as requisições de registro e de localização das chaves públicas.

A principal classe do protótipo de segurança é a classe *Security*. Esta classe abstrai o esquema inteiro de PKI e os mecanismos de segurança. A classe *Security* possui vários atributos:

[4]As bibliotecas do TSIK não contam com a implementação de um servidor XKMS, a *VeriSign* disponibiliza um servidor próprio para testes a serem realizados pelos usuários do TSIK. Várias tentativas de contactar a *VeriSign* para obter uma permissão para a utilização do seu servidor XKMS foram realizadas, mas nenhuma obteve sucesso. Sendo assim, tivemos que implementar um agente que simula um servidor XKMS para o nosso protótipo.

- Atributo *databasePKI* do tipo *prototype.security.databasePKI.DatabasePKI*: a classe *DatabasePKI* é responsável por representar uma base de dados PKI com várias chaves públicas estocadas. Quando a classe *Security* é utilizada pelo servidor XKMS, o atributo *databasePKI* contém as chaves públicas de outros agentes que foram registradas. Quando utilizado por um agente qualquer, o atributo *databasePKI* armazena as chaves públicas que já foram localizadas por este agente;

- Atributo *agent* tipo *jade.core.Agent*: representa o agente usuário da solução de segurança. A solução utiliza métodos do objeto *agent*, como o método *send* para enviar mensagens ACL;

- Atributo *xkmsServer* do tipo *jade.core.AID*: representa a identificação do agente responsável por simular um servidor XKMS. Este atributo representa o remetente de uma requisição de registro ou de localização de chave pública a ser enviada ao servidor XKMS, ou seja, o próprio servidor XKMS;

- Atributo *dhKeyPair* do tipo *java.security.KeyPair*: representa o par de valores público e privado utilizado no algoritmo do *Diffie-Hellman*;

- Atributo *sharedKey* do tipo *javax.crypto.SecretKey*: representa a chave secreta obtida pelo algoritmo do *Diffie-Hellman*. Esta chave corresponde a uma chave DES;

- Atributo *rsaKeyPair* do tipo *java.security.KeyPair*: representa o par de chaves pública e privada que participaram dos processos de criptografia e assinatura digital das mensagens seguras. Estas chaves são geradas pelo algoritmo RSA;

- Atributo *databasePKI* do tipo *prototype.security.databasePKI.DatabasePKI*: representa uma base de dados PKI de armazenamento de chaves públicas RSA;

- Atributo *xkmsServerOntology* do tipo *jade.content.onto.Ontology*: representa a ontologia utilizada pelos agentes e pelo servidor XKMS para a criação das requisições de registro e de localização de chave pública;

- Atributo *codec* do tipo *jade.content.lang.Codec*: representa o *codec* responsável pelo suporte ao padrão RDF. Este atributo na verdade contém uma instância de um objeto do tipo *jade.content.lang.Code.rdf.RDFCodec*.

A classe *Security* possui somente 6 (seis) métodos públicos que necessitam ser conhecidos e utilizados pelo usuário do protótipo:

- *public Security()*: método construtor da classe;
- *public void setAgent(Agent agent)*: define o agente usuário do protótipo que terá suas mensagens protegidas;
- *public void setXkmsServer(AID xkmsServer)*: armazena o AID do agente que simula o servidor XKMS;
- *public void setup()*: método responsável por gerar o par de chaves pública e privada RSA, gerar o par de valores do algoritmo do *Diffie-Hellman*, iniciar a sessão *Diffie-Hellman* com o servidor XKMS, registrar a chave pública do agente no servidor XKMS e receber a chave pública do servidor XKMS;
- *public void secureMessage(ACLMessage message)*: método responsável pela proteção de uma mensagem. Ele criptografa a mensagem utilizando a chave pública do receptor e assina a mensagem utilizando a chave privada do próprio remetente. Se o agente não possuir a chave pública do receptor, então ela é localizada;
- *public void recoverMessage(ACLMessage message)*: método responsável por recuperar uma mensagem de sua forma de texto cifrado para o texto normal. Ele verifica digitalmente a mensagem utilizando a chave pública do remetente e descriptografa a mensagem utilizando a sua próprio chave privada. Se a chave pública do remetente não for conhecida, então ela é localizada.

Os demais métodos da classe *Security* não necessitam ser conhecidos pelos usuários do protótipo, eles correspondem a dois métodos públicos e a todos os métodos privados da classe:

- *public void doKeyRegistration(ACLMessage message)*: método responsável por encapsular toda a complexidade de uma ação de registro de uma chave pública por parte do servidor XKMS;
- *public void doKeyLocation(ACLMessage message)*: método responsável por encapsular toda a complexidade de uma ação de localização de uma chave pública por parte do servidor XKMS;
- *private KeyPair generateRSAKeyPair()*: método responsável por gerar um par de chaves através do algoritmo RSA. Este par de chaves correspondem as chaves públicas e privadas dos agentes;

- *private KeyPair generateDHKeyPair()*: método responsável por gerar os valores públicos e privados do algoritmo do *Diffie-Hellman* para a criação de uma chave secreta entre os agentes e o servidor XKMS. Este método lê o arquivo de configuração para a criação dos valores do *Diffie-Hellman*;

- *private void setARPAddress()*: método responsável por ler o arquivo de configuração e definir no sistema operacional uma entrada estática correspondente ao endereço MAC do servidor XKMS;

- *private KeyResponse registerKey()*: método responsável por enviar uma requisição de registro da chave pública de um agente para o servidor XKMS. A mensagem é criptografada utilizando a chave secreta criada entre o agente e o servidor XKMS;

- *private KeyResponse locateKey(AID holder)*: método responsável por enviar uma requisição de localização de uma chave pública para o servidor XKMS. A mensagem é criptografada utilizando a chave pública do servidor XKMS;

- *private Key generateDESKey()*: método responsável por gerar uma chaves DES aleatória;

- *private void encryptMessage(ACLMessage message, Key DESKey, PublicKey receiverPublicKey)*: método responsável por criptografar uma mensagem destinada a um agente utilizando uma chave DES aleatória e a chave pública RSA do agente receptor;

- *private void signMessage(ACLMessage message)*: método responsável por assinar digitalmente uma mensagem utilizando a chave privada do agente remetente;

- *private boolean verifyMessage(ACLMessage message, PublicKey senderPublicKey)*: método responsável por verificar digitalmente uma mensagem utilizando a chave pública do agente rementente;

- *private void decryptMessage(ACLMessage message)*: método responsável por descriptografar uma mensagem utilizando a chave privada do próprio agente.

A Listagem 4.2 representa o uso da API de segurança por um agente que necessita enviar uma mensagem segura. O agente desse exemplo utiliza o *framework* JADE para que possa utilizar nossa API. O agente implementa uma classe chamada

SenderBehavior (que estende da classe *SimpleBehavior*) com um atributo do tipo *Security* que representa um objeto de segurança do nosso protótipo (linha 4). Nós instanciamos o objeto *security* na linha 7 e definimos os AIDs do agente que utiliza a API e do agente que simula um servidor XKMS (linha 8 e 9). Logo após, nós chamamos o método *setup* (linha 10). Antes de enviar a mensagem (linha 13), nós a tornamos segura utilizando o método *secureMessage*, passando a mensagem como argumento (linha 12).

Listagem 4.2: Código-fonte do agente *sender*

```
1  class Sender extends Agent {
2    [...]
3    class SenderBehaviour extends SimpleBehaviour {
4      private Security security;
5      [...]
6      public void action() {
7        security = new Security();
8        security.setAgent(myAgent);
9        security.setXkmsServer(new AID("xkmsServer", false));
10       security.setup();
11       [...]
12       security.secureMessage(message);
13       send(message);
14       [...]
15     }
16   }
17 }
```

A Figura 4.8 representa o diagrama de sequência da utilização do protótipo por parte do agente que envia uma mensagem segura. Nesta figura, somente a atividade do método *setup()* e a atividade do método *secureMessage(ACLMessage message)* estão representadas. O método *setup* é responsável pela execução do métodos *generateRSAKeyPair*, *generateDHKeyPair*, *setARPAddress*, *registerKey* e pela execução do método *add* do objeto *databasePKI* para o armazenamento da chave pública do servidor XKMS. O método *encryptMessage* é responsável por verificar se a chave pública do agente receptor já se encontra de posse do próprio agente (chamada do método *find* do objeto *databasePKI*), caso o agente ainda não tenha localizado a chave pública do agente receptor, ele poderá localizar a chave pública (método *locateKey*) e armazená-la (método *add* do objeto *databasePKI*). Uma chave DES aleatória é gerada (método

generateDESKey) e utilizada junto com a chave pública do agente receptor para a criptografia da mensagem (método *encryptMessage*). Após ser criptografada, a mensagem pode ser assinada digitalmente (método *signMessage*).

Figura 4.8: Diagrama de sequência do agente *sender*

A Listagem 4.3 apresenta o uso de nossa API de segurança por meio de um agente que precisa receber e processar uma mensagem segura. O agente implementa uma classe chamada *ReceiverBehavior* (que estende da classe *SimpleBehavior*) com um atributo que representa uma referência a classe *Security* de nosso protótipo (linha 4). Nós instanciamos o objeto *security* na linha 7 e definimos os AIDs do próprio agente e do agente que simula um servidor XKMS (linhas 8 e 9). Logo após isso, nós chamamos o método *setup* (linha 10). Após receber a mensagem na linha 12, nós podemos recuperar a mensagem de volta a sua forma normal utilizando o método *recoverMessage* passando a mensagem recebida como parâmetro (linha 13).

Listagem 4.3: Código-fonte do agente *receiver*

```
1  class Receiver extends Agent {
2    [...]
3    class ReceiverBehaviour extends SimpleBehaviour {
4      private Security security;
5      [...]
6      public void action() {
7        security = new Security();
8        security.setAgent(myAgent);
9        security.setXkmsServer(new AID("xkmsServer", false));
10       security.setup();
11       [...]
12       message = blockingReceive();
13       security.recoverMessage(message);
14       [...]
15     }
16   }
17 }
```

A Figura 4.9 representa o diagrama de sequência da utilização do protótipo por parte do agente *receiver*. Nesta figura, somente a atividade do método *recoverMessage(ACLMessage message)* está representada. O método *recoverMessage* é responsável por verificar se a chave pública do agente remetente já se encontra de posse do próprio agente (chamada do método *find* do objeto *databasePKI*), caso o agente ainda não tenha localizado a chave pública do agente receptor, ele poderá localizar a chave pública (método *locateKey*) e armazená-la (método *add* do objeto *databasePKI*). Já de posse da chave pública do agente remetente, podemos verificar digitalmente a mensagem (método *verifyMessage*) e descriptografá-la (método *decryptMessage*).

A Listagem 4.4 ilustra uma mensagem que foi submetida ao processo de segurança de nosso protótipo. Na mensagem, nós podemos notar a versão da especificação *XML-Enc* utilizada (linha 2), o algoritmo utilizado para criptografar o conteúdo da mensagem (linha 3), o algoritmo utilizado para criptografar a chave DES (linha 6), a própria chave DES criptografada (linha 8), o conteúdo da mensagem criptografado (linha 13), a versão da *XML-DSig* utilizada (linha 15), o algoritmo utilizado para a assinatura digital (linha 18), o código da assinatura digital (linha 28), a versão do algoritmo do sumário de mensagem (linha 24) e o resultado do sumário de mensagem (linha 28).

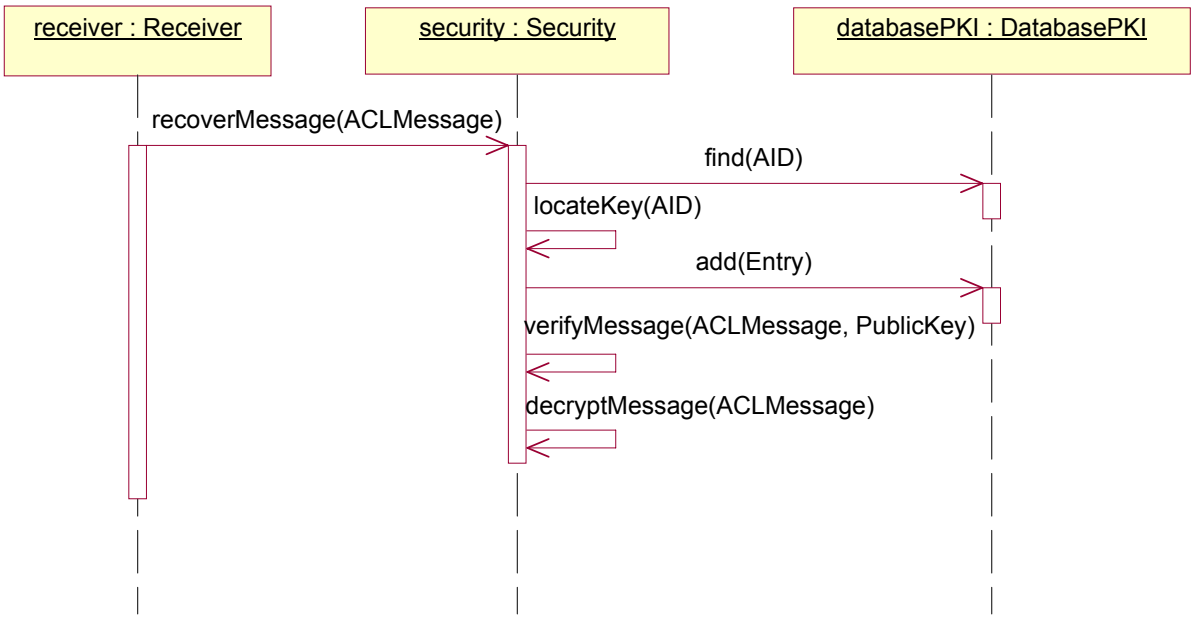


Figura 4.9: Diagrama de sequência do agente *receiver*

Listagem 4.4: Mensagem de exemplo da solução de segurança

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
       xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
3    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#
         tripledes-cbc"/>
4    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
5      <xenc:EncryptedKey>
6        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc
             #rsa-1_5"/>
7        <xenc:CipherData>
8          <xenc:CipherValue>mRlciAgKCBhZ2VudC1pZGVudGlm[...]</xenc:
               CipherValue>
9        </xenc:CipherData>
10     </xenc:EncryptedKey>
11   </ds:KeyInfo>
12   <xenc:CipherData>
13     <xenc:CipherValue>Yc4Tk+Eov7ox5hE1C8baQiZ96sEom[...]</xenc:
           CipherValue>
14   </xenc:CipherData>
15   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
16     <ds:SignedInfo>
17       <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/
             REC-xml-c14n-20010315"/>
```

```
18        <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
              rsa-sha1"/>
19        <ds:Reference URI="">
20          <ds:Transforms>
21            <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#
                  enveloped-signature"/>
22            <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-
                  c14n-20010315"/>
23          </ds:Transforms>
24          <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
                sha1"/>
25          <ds:DigestValue>79foHiQK0U/rcsJzsH0rDDQWnMc=</ds:DigestValue>
26        </ds:Reference>
27      </ds:SignedInfo>
28      <ds:SignatureValue>Wm+BfwE1R7xyIz4WVf4vTPvqWsL[...]</ds:
            SignatureValue>
29    </ds:Signature>
30  </xenc:EncryptedData>
```

4.7 Comparativo entre a Solução de Segurança e o JADE-S

Comparando o funcionamento da solução de segurança proposta e o JADE-S, nós podemos observar que:

- O JADE-S faz as requisições das chaves públicas aos próprios agentes e não possui um agente centralizador, como o servidor XKMS;
- O JADE-S funciona somente para agentes em uma mesma plataforma, enquanto que na solução proposta de segurança temos comunicação segura entre plataformas diferentes;
- A utilização do JADE-S possui uma complexidade de codificação muito maior em relação a utilização da nossa solução de segurança;
- A solução de segurança por utilizar especificações XML abertas possui interoperabilidade, já que ela pode registrar e localizar chaves públicas em servidores

XKMS externos, se necessário.

4.8 Conclusões

A solução de comunicação segura proposta realmente fornece mecanismos de autenticação, confidenciabilidade, não-repúdio do remetente e integridade para sistemas multiagentes. A confidenciabilidade é obtida pela criptografia, a autenticação e o não-repúdio são obtidos pela assinatura digital e a integridade é obtida pelo sumário de mensagem.

Nós adaptamos a especificação XKMS para fornecer um esquema PKI para o modelo proposto, fornecemos confidenciabilidade através da especificação *XML Encryption* e implementamos assinaturas digitais através da especificação *XML Signature*. Todas as implementações foram adaptadas a partir do TSIK, um *toolkit* disponível pela *VeriSign*.

A solução proposta usa também um *codec* para habilitar o suporte RDF para o agentes.

O protótipo fornece interfaces amigáveis que abstraem toda a complexidade dos mecanismos de segurança. O protótipo pode ser utilizado entre diferentes plataformas JADE. Nosso protótipo também possui uma complexidade de codificação muito menor do que a complexidade de codificação utilizando o JADE-S.

A vantagem de se adaptar especificações XML é a interoperabilidade, pois os sistemas multiagentes que utilizarem a solução de segurança proposta podem interagir com outros sistemas que utilizem as mesmas especificações.

Comparando a solução de segurança proposta e o JADE-S, nós podemos observar que nossa solução mostrou a vantagem de poder ser utilizada em plataformas JADE diferentes.

5 Modelagem e Implementação da Solução de Confiabilidade

Este capítulo tem por objetivo apresentar em um nível de detalhes mais aprofundado a solução de comunicação confiável proposta nesta dissertação. Neste capítulo, apresentamos: a visão geral da solução de confiabilidade; um pequeno manual prático de como implementar um MTP para o JADE; a modelagem e prototipagem da solução; a utilização da solução com um exemplo para o JADE; um comparativo da solução com o JMS-MTP; e, finalmente, apresentamos as conclusões acerca da solução de confiabilidade proposta.

5.1 Visão Geral da Solução de Confiabilidade

A solução de confiabilidade proposta nesta dissertação adapta os protocolos de confirmação de recebimento de mensagens fornecidos pela especificação WS-RM [2]. Assim, os protocolos do WS-RM são adaptados para sistemas multiagentes que suportem o padrão RDF.

A solução teve seu protótipo implementado em JAVA como um MTP para o *framework* JADE e, claro, pode ser utilizado por qualquer um que use este *framework*.

5.2 Pequeno Manual Prático de Implementação de um MTP-JADE

Após a decisão de implementarmos um protótipo da solução de garantia de entrega de mensagens como um MTP, iniciamos a coleta de informações e conhecimento para atingirmos este objetivo. Descobrimos que não havia nenhum tutorial ou algo que o valha como tal para auxiliar na implementação de um MTP, então esta dissertação apresenta um pequeno manual prático (ou *"how-to"*) de como implementar um MTP-JADE.

Podemos dizer que esta seção tem por objetivo apresentar as informações de como lançar rapidamente e sem muitos detalhes seu próprio protótipo de MTP. Esta seção também serve para apresentar considerações sobre um MTP-JADE necessários para o entendimento da modelagem do protótipo.

5.2.1 Interface MTP

O ponto inicial da implementação de um MTP é a implementação da interface *MTP* do pacote *jade.mtp*. Os métodos dos canais de entrada e saída de um MTP encontram-se em duas outras interfaces, a *InChannel* e a *OutChannel*. A interface *MTP* estende estas duas classes.

Assim, a interface *MTP* contém realmente métodos que devem ser implementados para a criação de um MTP:

- *public String[] getSupportedProtocols()*: retorna um vetor de *Strings*, cada elemento do vetor corresponde a um protocolo TCP/IP suportado pelo MTP. Exemplo: {*"http", "https"*};
- *public String getName()*: retorna o nome do MTP registrado junto a FIPA. Em um protótipo, pode assumir uma constante vazia ou um valor qualquer. Exemplo de um nome de um MTP já registrada: *"fipa.mts.mtp.http.std"*;
- *public TransportAddress strToAddr(String rep)*: retorna um objeto do tipo *TransportAddress* correspondente à representação em *String* de um endereço de transporte válido para este MTP informada como parâmetro. Um objeto do tipo *TransportAddress* representa um endereço válido para um MTP e será explicado na próxima seção;
- *public String addrToStr(TransportAddress ta)*: retorna uma representação em *String* correspondente à de um endereço de transporte válido para este MTP informada como parâmetro de objeto do tipo *TransportAddress*. Exemplo: *"https://myhost:7778/acc"*;
- *public TransportAddress activate(Dispatcher disp, Profile p)*: método responsável por ativar o mecanismo de recebimento de mensagens do próprio MTP. O objeto *disp* do tipo *InChannel.Dispacther* recebido como parâmetro tem um papel importante

para o MTP, será o objeto *disp* que através de seu método *dispatchMessage*, irá inserir na fila de mensagens de uma plataforma de agentes, uma mensagem recebida. O parâmetro tipo *jade.core.Profile* [19] armazena as propriedades do próprio agente. O método ainda retorna um objeto do tipo *TransportAddress* que representa o endereço em que o MTP foi ativado;

- *public void deactivate()*: desativa o mecanismo de recebimento de mensagens do MTP ativado no endereço padrão da máquina;

- *public void deliver(String addr, Envelope env, byte[] payload)*: este método é responsável pela entrega de uma mensagen ACL. Resumidamente, o papel do método *deliver* é fazer com que o objeto do tipo *Envelope* e o vetor de *bytes* recebidos como parâmetros sejam enviados para o agente que possuir o mesmo endereço de transporte representado pelo parâmetro *String addr*. Um objeto do tipo *Envelope* representa o cabeçalho de uma mensagem ACL, contendo campos referentes ao receptor da mensagem e a codificação da mensagem. O vetor de *bytes* representa o corpo da mensagem submetido a uma codificação. Os dois parâmetros juntos representam uma mensagem ACL completa com corpo e cabeçalho. Resumidamente, este método é responsável pela ativação do mecanismo de envio de uma mensagem para outro agente que utilize o mesmo MTP;

- *public void activate(Dispatcher disp, TransportAddress ta, Profile p)*: este método é semelhante ao método *public TransportAddress activate(Dispatcher disp, Profile p)*, exceto que ele ativa o MTP para responder no endereço de transporte informada como parâmetro. Este método não necessariamente precisa ser implementado em uma versão de protótipo;

- *public void deactivate(TransportAddress ta)*: Este método é semelhante ao método *public void deactivate()*, com exceção que ele desativa o MTP que esteja respondendo pelo endereço de transporte informado como parâmetro. Este método não necessariamente precisa ser implementado em uma versão de protótipo.

A implementação da interface *MTP* pode conter também a implementação do mecanismo de troca de mensagens e os métodos referentes ao mesmo.

5.2.2 Interface *TransportAddress*

A interface *TransportAddress* é bastante simples e conta somente com os métodos *get* das propriedades de um endereço de transporte de um MTP. As propriedades de um endereço de transporte assemelham-se muito às propriedades de uma URL.

- *String getProto()*: retorna o protocolo de um endereço de transporte. Exemplo: *"https"* de *"https://myhost:7778/acc"*;
- *String getHost()*: retorna o nome de rede de um endereço de transporte. Exemplo: *"myhost"* de *"https://myhost:7778/acc"*;
- *String getPort()*: retorna a porta de um endereço de transporte. Exemplo: *"7778"* de *"https://myhost:7778/acc"*;
- *String getFile()*: retorna o nome de arquivo de um endereço de transporte. Exemplo: *"/acc"* de *"https://myhost:7778/acc"*;
- *String getAnchor()*: retorna o nome da âncora (ou referência) de um endereço de transporte. Esta propriedade é rara de se encontrar em um endereço de transporte de um MTP. Exemplo: *"ancora"* de *"https://myhost:7778/acc#ancora"*.

5.2.3 Implementando seu Próprio MTP

Já que as interfaces a serem implementadas já foram apresentadas, podemos resumir que uma implementação rápida de um MTP consiste em:

- Implementar o método *public String[] getSupportedProtocols()* da interface *MTP*: fazendo com que este retorne o vetor de *Strings* correspondentes aos protocolos TCP/IP suportados pelo MTP. Exemplo: *"http"*, *"https"*;
- Implementar o método *public String getName()* da interface *MTP*: fazendo com que este retorne o nome do MTP registrado junto a FIPA ou um mero valor arbitrário;
- Implementar o método *public TransportAddress strToAddr(String rep)* da interface *MTP*: fazendo com que este retorne um objeto do tipo *TransportAddress* correspondente à representação em *String* de um endereço de transporte válido passado como parâmetro. Pode-se atingir este objetivo, implementando um

construtor para a classe *TransportAddress* que aceite a representação *String* como parâmetro;

- Tratar e gerar exceções *MTPException*: todos os métodos da interface *MTP*, com exceção de *public String[] getSupportedProtocols()* e *public String getName()*, retornam eventualmente uma exceção do tipo *MTPException*. Uma maneira rápida de fazer com que suas implementações gerem esta exceção é ilustrada na Listagem 5.1;

Listagem 5.1: Tratando Exceções *MTPException*

```
1 public TransportAddress strToAddr(String rep) throws MTPException {
2         try {
3                 [...]
4         } catch (Exception e) {
5                 throw new MTPException(e.toString());
6         }
7 }
```

- Implementar o método *public String addrToStr(TransportAddress ta)* da interface *MTP*: fazendo com que este retorne uma representação em *String* correspondente ao do parâmetro do tipo *TransportAddress*. Pode-se atingir este objetivo, implementando o método *toString()* para a classe *TransportAddress*;

- Implementar o método *public TransportAddress activate(InChannel.Dispatcher disp, Profile p)* da interface *MTP*: fazendo com que este retorne um objeto do tipo *TransportAddress* que representa o endereço em que o MTP foi ativado; além de ativar o mecanismo de recebimento de mensagens do MTP. Exemplos: um *HTTP server* com métodos PUT habilitados, um web service, ou uma porta de comunicação qualquer aliada a um *daemon*[1] e a um processo de recebimento das mensagens. Este método também é responsável por armazenar a referência ao objeto do tipo *InChannel.Dispacther*, para utilizarmos o seu método *void dispatchMessage(Envelope env, byte[] payload)* e inserir um mensagem recebida na fila de mensagens de uma plataforma de agentes. O parâmetro *p* do tipo

[1]Um *daemon* corresponde a um pequeno programa de escuta em execução contínua que tem como tarefa executar um outro programa maior que é responsável pelo processamento de informações recebido pelo *daemon*. Assim que o programa maior termina de processar as informações ele é finalizado. Exemplo: um *daemon* de um *web server*.

jade.core.Profile [19] armazena as configurações do agente que podem ser utilizadas para a criação do objeto *TransportAddress*;

- Implementar o método *public void deactivate()* da interface *MTP*: fazendo com que este tenha o papel de desativar o mecanismo de transporte de mensagens do MTP ativado no endereço padrão da máquina;

- Implementar o método *public void deliver(String addr, Envelope env, byte[] payload)* da interface *MTP*: fazendo com que este utilize o mecanismo de recebimento de mensagens de um agente com um endereço de transporte representado pelo parâmetro *String addr* para enviar uma mensagen ACL. Exemplos: através de uma chamada do método PUT de um *HTTP server*, de uma chamada a um *web service*, ou pela conexão a uma porta pré-determinada que possua mecanismos de escuta e processamentos especializados do próprio MTP. Resumidamente, o papel do método *deliver* é fazer com que o objeto do tipo *Envelope* e o vetor de *bytes* recebidos como parâmetros sejam enviados para o agente que possui o mesmo endereço de transporte representado pela *String*. Um objeto do tipo *Envelope* representa o cabeçalho de uma mensagem ACL, contendo campos referentes ao receptor da mensagem e à codificação da mensagem. O vetor de *bytes* representa o corpo da mensagem submetido a uma codificação. Os dois parâmetros juntos representam uma mensagem ACL completa com corpo e cabeçalho;

- Implementar o seu próprio mecanismo de envio e de recebimento de mensagens: o mecanismo de transporte de mensagens pode ser implementado na interface *MTP* ou em classes separadas. Por exemplo, um mecanismo de recebimento de mensagens pode ser um *web service* e o mecanismo de envio pode ser uma chamada remota ao *web service*; ou um *HTTP server* com os métodos PUT habilitados, e as chamadas aos métodos PUT remotamente;

- Como já foi dito, o método *public void activate(Dispatcher disp, TransportAddress ta, Profile p)* e o método *public void deactivate(TransportAddress ta)* não necessariamente precisam ser implementados em uma versão de protótipo de um MTP;

- Implementar a interface *TransporteAddress*, implementando todos os seus métodos *get*, seu construtor e seu método *toString()*. Como um endereço de transporte

assemelha-se muito a uma URL, podemos utilizar uma classe da própria API do Java conhecida como *java.net.URL* [32]. A classe *URL* fornece construtores, métodos *get* e um método *toString()* que podem ser facilmente utilizados na implementação da interface *TransportAddress*.

Para finalizar, uma outra forma resumida de ver o funcionamento de um MTP é fazer com que os parâmetros *env* e *payload* do método *deliver(String addr, Envelope env, byte[] payload)* sejam entregues ao agente de endereço idêntico ao parâmetro *addr*; e que o agente receptor insira os objetos recebidos *env* e *payload* na fila de mensagens através do método *dispatchMessage(Envelope env, byte[] payload)* do objeto *disp*. Após ser recebida como parâmetro do método *activate(InChannel.Dispatcher disp, Profile p)*, a referência para o objeto *disp* deve ser devidamente armazenada.

5.3 Modelagem e Prototipagem

Esta seção tem por objetivo apresentar os detalhes a cerca da modelagem e prototipagem da solução de confiabilidade proposta nesta dissertação.

Para a implementação do protótipo utilizamos a API Java *Systinet Server for Java* da companhia Systinet [36]. A Systinet fornece produtos para a arquitetura SOA (*Service Oriented Architecture*) baseados em padrões como XML e SOAP.

O *Systinet Server for Java* trata-se de um ambiente independente de plataforma, de fácil utilização e de alto desempenho para a criação e desenvolvimento de *web services* em Java para aplicações J2EE.

Escolhemos o *Systinet Server* por ser uma solução completa e por possuir uma implementação da especificação WS-RM, que torna a implementação do protótipo muito ágil e fácil.

Resumidamente, o protótipo da solução de confiabilidade é um MTP-JADE escrito em Java utilizando um *web service*, onde as chamadas feitas ao *web service* são confirmadas através dos protocolos do WS-RM. As chamadas a um método do *web service* são utilizadas para enviar as mensagens de um agente para o outro.

5.3.1 Pacotes do Protótipo

O diagrama de pacotes do protótipo de confiabilidade está ilustrado na Figura 5.1. As classes do protótipo da solução de confiabilidade foram agrupadas no pacote *prototype.reliability* (Figura 5.1). O pacote *prototype.reliability* possui uma dependência do pacote *jade.mtp* por conter uma implementação das interfaces *MTP* e *TransportAddress* do pacote *jade.mpt*; possui uma dependência do pacote *jade.domain.FIPAAgentManagement* por manipular objetos do tipo *Envelope* do pacote *jade.domain.FIPAAgentManagement*; e possui uma dependência ao pacote *org.systinet.wasp* pela utilização dos componentes de *web services* e da especificação WS-RM do pacote *org.systinet.wasp*.

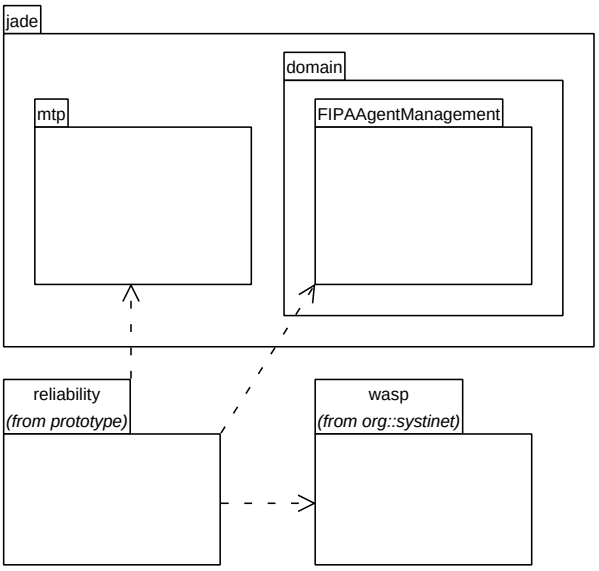


Figura 5.1: Diagrama de pacotes do protótipo da solução de confiabilidade

5.3.2 Classes do Protótipo

O diagrama de classes do protótipo de confiabilidade está ilustrado na Figura 5.2.

No protótipo, a classe *MessageTransportProtocol* é a classe que implementa a interface *MTP* e a classe *ReliableAddress* é a classe que implementa a interface *TransportAddress* (Figura 5.2).

A classe *ReliableAddress* representa um endereço de transporte válido para este MTP e utiliza a classe *java.net.URL* [32] para a implementação do construtor, métodos

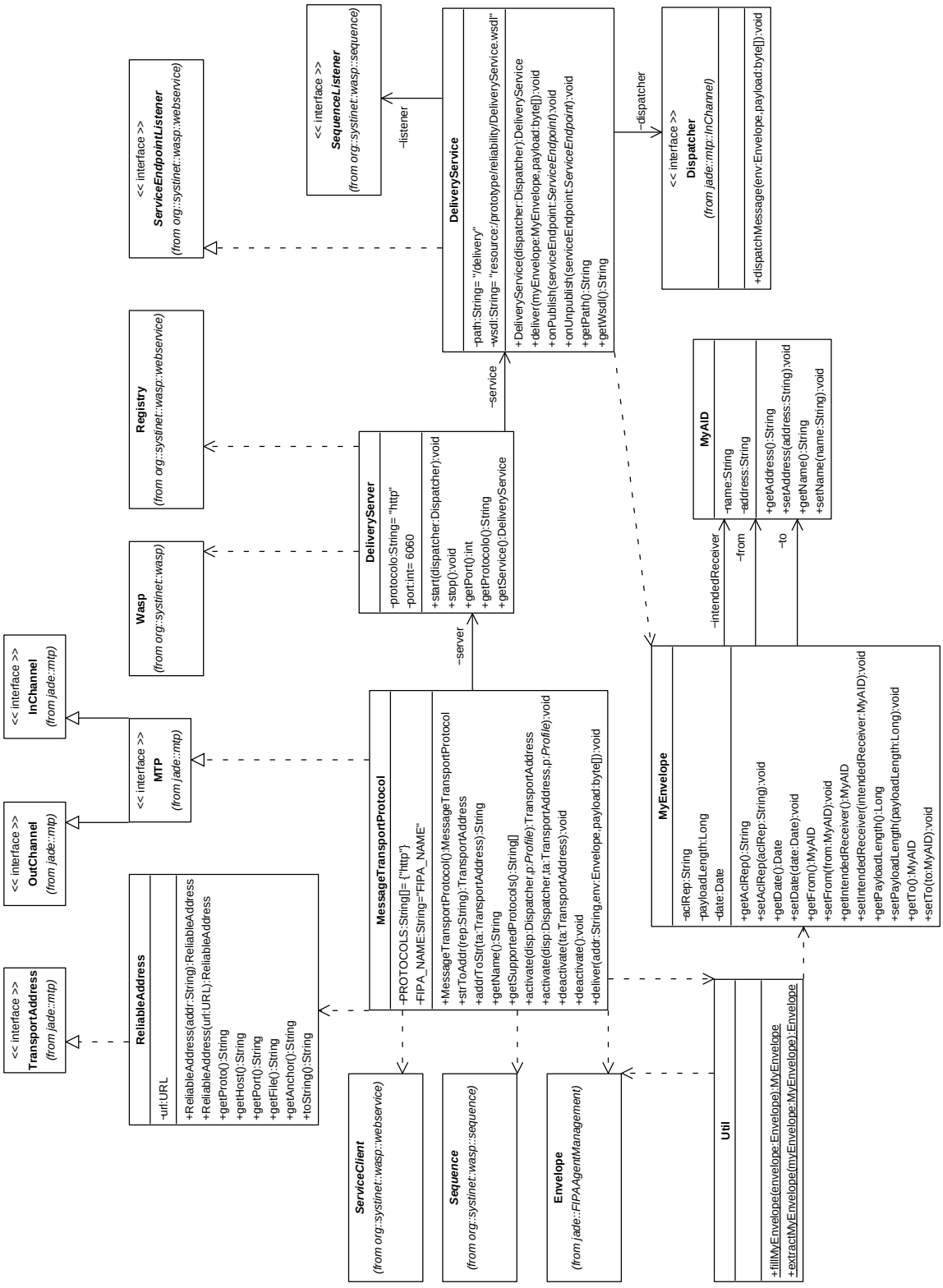


Figura 5.2: Diagrama de classes do protótipo da solução de confiabilidade

get e o método *toString* (vide o atributo *url* da classe *ReliableAddress* na Figura 5.2). A construtora da classe *ReliableAddress* é utilizada para a criação do objeto de acordo com a representação de endereço passada como parâmetro e o método *toString* é utilizado para retornar a representação de endereço. Exemplo de representação de endereço de

transporte válido para este MTP: *"http://myhost:6060/delivery"*.

Como o mecanismo de envio de mensagens é um *web service*, o que implica também em um *web server*, os protocolos TCP/IP suportados por este MTP correspondem somente ao HTTP (vide a constante atributo *PROTOCOLS* que é retornada pelo método *getSupportedProtocols* da classe *MessageTransportProtocolo* na Figura 5.2). O nome que deveria estar registrado para este MTP na FIPA trata-se somente do valor arbitrário *"FIPA_NAME"* (vide a constante atributo *FIPA_NAME* que é retornada pelo método *getName* da classe *MessageTransportProtocolo* na Figura 5.2).

O método *strToAddr* retorna um objeto do tipo *ReliableAddress* de acordo com a representação de endereço em *String* informada e utiliza a construtora da classe *ReliableAddress* para tanto.

O método *addrToStr* retorna uma representação de endereço em *String* correspondente ao objeto do tipo *ReliableAddress* informado e utiliza o método *toString* da classe *ReliableAddress* para tanto.

A construtora da classe *MessageTransportProtocol* é utilizada para criar uma instância da classe *DeliveryServer* em seu atributo *server*.

A classe *DeliveryServer* representa um servidor *web* que suporta o *web service* do mecanismo de envio de mensagens. Esta classe possui dois atributos, o atributo *protocol* e o atributo *port* que representam o protocolo suportado e a porta do servidor *web*, respectivamente. No caso deste MTP, o servidor *web* escutará pela porta 6060.

A classe *DeliveryServer* possui dois métodos responsáveis por ativar e desativar o próprio servidor *web*, respectivamente o método *start* e o método *stop*. Estes métodos utilizam os métodos da classe estática *org.systinet.wasp.Wasp* [37] para ativar e desativar o servidor *web*.

O método *start* da classe *DeliveryServer* é responsável ainda por instanciar um objeto do tipo *DeliveryService* e associá-lo ao atributo *service* da mesma classe. A classe *DeliveryService* representa o *web service* utilizado por este MTP. O atributo *service* é registrado no *web server* através do método *publish* da classe estática *org.systinet.wasp.webservice.Registry* [37].

O método *start* da classe *DeliveryServer* recebe como parâmetro um objeto do tipo *InChannel.Dispatcher* que é repassado para o construtor da classe *DeliveryService*.

De posse de um objeto do tipo *InChannel.Dispatcher*, a classe *DeliveryService* poderá incluir uma mensagem recebida na fila de mensagens.

A classe *DeliveryService* possui dois atributos, o atributo *path* e o atributo *wsdl*, que representam respectivamente, o caminho do registro do método *deliver* do *web service* no servidor *web*, e o caminho do arquivo *wsdl* do *web service*. Como a porta do servidor *web* é a 6060 e o caminho do registro do método é o *"/delivery"*, as representações de endereços válidas deste MTP são semelhantes a este: *"http://myhost:6060/delivery"*.

O construtor da classe *DeliveryService* serve para armazenar a referência do objeto do tipo *InChannel.Dispatcher* no atributo *dispatcher*.

A classe *DeliveryService* possui ainda uma atributo *listener* do tipo *org.systinet.wasp.sequence.SequenceListener* [37] que é utilizado para o monitoramento das alterações de estado de uma seqüência de mensagens do WS-RM. Os métodos *onPublish* e *onUnpublish* servem, respectivamente, para adicionar e remover o atributo *listener*.

O método *deliver* da classe *DeliveryService* representa o mecanismo de recebimento de mensagens deste MTP.

Voltando para a classe *MessageTransportProtocol*, o método *activate(Dispatcher disp, Profile p)* é responsável por ativar o servidor *web* através da chamada do método *start* do atributo *server*, passando como parâmetro o objeto *disp* do tipo *InChannel.Dispatcher*. Como já foi mostrado, a instância da classe *DeliveryServer* será responsável por criar e registrar o *web service* representado pelo seu atributo *service* do tipo *DeliveryService* e por repassar o objeto do tipo *InChannel.Dispatcher* para o seu atributo *service*.

O método *deactivate()* da classe *MessageTransportProtocol* é responsável pela chamada do método *stop* do atributo *server*, realizando a desativação do *web server*.

Os métodos *activate(Dispatcher disp, TransportAddress ta, Profile p)* e *deactivate(TransportAddress ta)* não estão realmente implementados neste protótipo.

O método *deliver* da classe *MessageTransportProtocol* representa o mecanismo de envio de uma mensagem a outro agente. É de responsabilidade do método *deliver* realizar um envio de uma mensagem ACL através de uma chamada do método *deliver* da classe *DeliveryService*. O método *deliver* da classe *DeliveryService*, como já foi dito, representa o mecanismo de recebimento de mensagens deste MTP. É assim, então, que se

dá o mecanismo de transporte de mensagens neste MTP, através de uma chamada de um método de um *web service* passando como parâmetro uma mensagem ACL.

O método *deliver* da classe *MessageTransporProtocol* recebe como parâmetro uma representação de um endereço válido deste MTP[2]. Esta representação é utilizada para o reconhecimento do agente para onde a mensagem será enviada. Uma mensagem ACL é representada pelos dois outros parâmetros recebidos, o objeto *env* do tipo *Envelope* e o objeto *payload*, um *array* de *bytes*.

Um objeto do tipo *Envelope* não pode ser utilizado como parâmetro de um método de um *web service*, por não estar de acordo com as especificações da arquitetura JAXB [33]. Então, para enviar um objeto do tipo *Envelope* ao *web service* é criado um objeto do tipo *MyEnvelope* que atende a arquitetura JAXB e que contem as mesmas informações do objeto *Envelope*. Um método estático *fillMyEnvelope* da classe *Util* do mesmo pacote é responsável pela criação de um objeto do tipo *MyEnvelope* de acordo com o objeto *Envelope* passado como parâmetro.

O método *deliver* da classe *MessageTransportProtocol* então cria um objeto do tipo *MyEnvelope* para ser passado como parâmetro do método *deliver* da classe *DeliveryService* junto com o *array* de *bytes* que representa o *payload* da mensagem. A chamada do método do *web service* é realizado através dos métodos da classe *org.systinet.wasp.webservice.ServiceClient* [37]. Uma seqüência (do WS-RM) de uma única posição também é criada para a chamada do método do *web service*. Ao realizar a chamada ao método do *web service* remoto, o agente finaliza o mecanismo de envio de uma mensagem.

No agente remoto, o método *deliver* da classe *DeliveryService* então é chamado e recebe como parâmetros um objeto do tipo *MyEnvelope* e um vetor de *bytes* que representa o *payload* da mensagem. Um objeto do tipo *Envelope* é criado apartir do objeto *MyEnvelope* recebido. Um método estático *extractMyEnvelope* da classe *Util* do mesmo pacote é responsável pela criação de um objeto do tipo *Envelope* a partir de um objeto *MyEnvelope* passado como parâmetro. O método então pode utilizar o método *dispatchMessage* do seu atributo *dispatcher* do tipo *InChannel.Dispatcher* passando o objeto *Envelope* e o vetor de *bytes*, inserindo, finalmente, a mensagem ACL na fila de mensagens. Este procedimento finaliza o mecanismo de recebimento de mensagens deste

[2]Exemplo de um endereço válido: *"http://myhost:6060/delivery"*.

MTP.

5.4 Exemplos

Para executarmos um exemplo utilizando o protótipo da solução de confiabilidade, devemos criar uma variável de ambiente chamada WASP_HOME que contenha o diretório onde a API e a biblioteca do *Systinet* estejam instalados. Exemplo:

$$WASP\_HOME = C:\backslash systinet\_server\_java60\backslash lib.$$

Devemos também incluir as seguintes entradas na variável de ambiente CLASSPATH:

$$\%WASP\_HOME\%\backslash lib\backslash wasp.jar; \%WASP\_HOME\%\backslash lib\backslash ws\_rm\_client.jar.$$

Ao executarmos o JADE pela linha de comando devemos incluir uma propriedade de configuração, a

$$-Dwasp\_home = \%WASP\_HOME\%.$$

Devemos também informar que o nosso MTP deve ser utilizado, como é mostrado a seguir:

$$-mtp\ prototype.reliability.MessageTransportProtocol.$$

Assim, a linha de comando de execução do JADE finalmente está representada na Listagem 5.2.

Listagem 5.2: Linha de execução do protótipo de confiabilidade

```
1 java -Dwasp_home=%WASP_HOME% jade.Boot -mtp prototype.reliability.
    MessageTransportProtocol sender:prototype.samples.Sender
```

A classe *prototype.samples.Sender* é responsável por enviar uma mensagem para outro agente chamado *receiver*.

Podemos executar o agente *receiver* em outra máquina igualmente configurada através do comando representado na Listagem 5.3.

Listagem 5.3: Linha de execução do agente *receiver*

```
1 java -Dwasp_home=%WASP_HOME% jade.Boot -mtp prototype.reliability.
    MessageTransportProtocol receiver:prototype.samples.Receiver
```

No código-fonte do agente *sender* devemos incluir o endereço de transporte do agente *receiver*, como mostrado na Listagem 5.4.

Listagem 5.4: Incluindo um endereço de transporte a um agente

```
1 receiver = new AID("receiver@pc01:1099/JADE", AID.ISGUID);
2 receiver.addAddresses("http://receiver:6060/delivery")
```

A Listagem 5.5 representa uma mensagem enviada para o agente *receiver*. As linhas de 2 a 10 da Listagem 5.5 representam as informações de cabeçalho do WS-RM. O cabeçalho informa ao agente *receiver* que a mensagem recebida pertence a uma seqüência com somente uma posição (uma seqüência composta por somente uma mensagem). As linhas de 12 a 30 representam os parâmetros do método *deliver* do *web service*. As linhas 13 a 29 representam as informações do envelope da mensagem ACL convertidas para a classe *MyEnvelope*. A linha 30 representa o conteúdo da mensagem codificada em um *array* de *bytes*.

Listagem 5.5: Mensagem do protótipo da solução de confiabilidade

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <e:Envelope xmlns:d="http://www.w3.org/2001/XMLSchema" xmlns:i="http://
    www.w3.org/2001/XMLSchema-instance" xmlns:wsrm="http://schemas.
    xmlsoap.org/ws/2003/03/rm" xmlns:wsu="http://schemas.xmlsoap.org/ws
    /2002/07/utility" xmlns:e="http://schemas.xmlsoap.org/soap/envelope
    /">
3   <e:Header>
4     <wsrm:Sequence e:mustUnderstand="1">
5       <wsu:Identifier>urn:6e94c052-a3a3-11da-afbd-6e94c051afbd</wsu:
          Identifier>
6       <wsrm:MessageNumber>1</wsrm:MessageNumber>
7       <wsrm:LastMessage />
8       <wsu:Expires>2007-02-22T10:02:09.109-03:00</wsu:Expires>
9     </wsrm:Sequence>
10  </e:Header>
11  <e:Body>
12    <ns2:send xmlns:ns0="http://systinet.com/wsdl/model/reliability/"
        xmlns:ns2="http://systinet.com/wsdl/model/reliability/">
```

```
13        <ns2:p0 i:type="ns0:MyEnvelope">
14          <ns2:aclRep i:type="d:string">fipa.acl.rep.string.std</ns2:
               aclRep>
15          <ns2:date i:type="d:dateTime">2006-02-22T10:02:09.015-03:00</ns2
               :date>
16          <ns2:from i:type="ns0:MyAID">
17            <ns2:address i:type="d:string">http://192.168.4.13:6060</ns2:
                 address>
18            <ns2:name i:type="d:string">receiver@pc01:1099/JADE</ns2:name>
19          </ns2:from>
20          <ns2:intendedReceiver i:type="ns0:MyAID">
21            <ns2:address i:type="d:string">http://192.168.4.13:6060</ns2:
                 address>
22            <ns2:name i:type="d:string">receiver@pc01:1099/JADE</ns2:name>
23          </ns2:intendedReceiver>
24          <ns2:payloadLength i:type="d:long">1350</ns2:payloadLength>
25          <ns2:to i:type="ns0:MyAID">
26            <ns2:address i:type="d:string">http://192.168.4.13:6060</ns2:
                 address>
27            <ns2:name i:type="d:string">receiver@pc01:1099/JADE</ns2:name>
28          </ns2:to>
29        </ns2:p0>
30        <ns1:p1 i:type="d:base64Binary" xmlns:ns1="http://systinet.com/
             wsdl/model/reliability/">
             KElORk9STQogOnNlbmRlciAgKCBhZ2VudC1pZGVudGlma[...]</ns1:p1>
31      </ns2:send>
32    </e:Body>
33  </e:Envelope>
```

A Listagem 5.6 representa a mensagem de confirmação recebida pelo agente *sender*. A linha 6 da Listagem 5.6 informa ao agente *sender* que todas as mensagens da seqüência foram recebidas com sucesso. No nosso caso, a única mensagem da seqüência foi recebida com sucesso.

Listagem 5.6: Mensagem de confirmação do protótipo da solução de confiabilidade

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <e:Envelope xmlns:d="http://www.w3.org/2001/XMLSchema" xmlns:i="http://
     www.w3.org/2001/XMLSchema-instance" xmlns:wsrm="http://schemas.
     xmlsoap.org/ws/2003/03/rm" xmlns:wsu="http://schemas.xmlsoap.org/ws
     /2002/07/utility" xmlns:e="http://schemas.xmlsoap.org/soap/envelope
```

```
    /">
3   <e:Header>
4     <wsrm:SequenceAcknowledgement e:mustUnderstand="1">
5       <wsu:Identifier>urn:6e94c052-a3a3-11da-afbd-6e94c051afbd</wsu:
            Identifier>
6       <wsrm:AcknowledgementRange Upper="1" Lower="1" />
7     </wsrm:SequenceAcknowledgement>
8   </e:Header>
9   <e:Body>
10    <ns0:sendResponse xmlns:ns0="http://systinet.com/wsdl/model/
          reliability/" />
11  </e:Body>
12 </e:Envelope>
```

5.5 Comparativo entre a Solução de Confiabilidade e o JMS-MTP

Comparando o funcionamento da solução de confiabilidade proposta e o JMS-MTP, nós podemos observar que:

- O JMS-MTP necessita de um servidor JMS para a criação de filas que contêm as mensagens enviadas de uma plataforma JADE para outra. A solução de confiabilidade proposta utiliza um *web service* em cada máquina para o envio das mensagens;
- O JMS-MTP periodicamente acessa o servidor JMS para verificar se há novas mensagens para a sua plataforma. A solução de confiabilidade proposta realiza as entregas de mensagens diretamente aos agentes;

5.6 Conclusões

A solução de comunicação confiável proposta realmente fornece mecanismos de garantia de entrega de mensagens para sistemas multiagentes.

Nós adaptamos a especificação WS-RM para fornecer mecanismos de garantia de entrega de mensagens por meio dos protocolos de confirmações da WS-RM.

Neste capítulo, nós apresentamos um pequeno manual prático de como implementar um MTP-JADE que ilustra os passos de como implementar rapidamente um MTP e que serve de base para a apresentação do nosso próprio protótipo.

O protótipo foi implementado como um MTP-JADE que o torna relativamente transparente aos usuários do JADE. Uma API disponível pela *Sysinet* com a implementação da WS-RM foi utilizada para o desenvolvimento rápido do protótipo da solução de confiabilidade.

Comparando a solução de confiabilidade proposta e o JMS-MTP, nós podemos observar que nossa solução envia as mensagens de forma direta aos agentes por meio de um *web service*, enquanto que o JMS-MTP utiliza um servidor JMS, fazendo com que os agentes não recebam suas mensagens diretamente e tenham que esperar por consultas periódicas ao servidor JMS para receberem de fato suas mensagens.

6 IDS NIDIA Como Estudo de Caso

Este capítulo apresenta o Projeto NIDIA (*Network Intrusion Detection System based on Intelligent Agents*) e ilustra a sua utilização como estudo de caso dos protótipos da solução de segurança e de confiabilidade. A arquitetura e as principais características do NIDIA também são apresentadas.

6.1 Visão Geral do NIDIA

O Projeto ***N**etwork **I**ntrusion **D**etection System based on **I**ntelligent **A**gents* (NIDIA) [23] é um sistema de detecção de intrusão baseado no conceito de sociedade de agentes inteligentes. O IDS NIDIA é capaz de detectar novos ataques através de uma rede neural, em tempo real. O NIDIA foi idealizado com o propósito de fornecer uma contribuição para a melhoria das técnicas de detecção de intrusos em redes de computadores com a utilização de agentes inteligentes e redes neurais em mecanismos de reconhecimento de ataques [6], associando, também, capacidades reativas [4].

Resumidamente, o monitoramento utiliza uma combinação de agentes sensores instalados nos pontos estratégicos da rede e nos *hosts* da rede. Estes agentes possuem o objetivo de capturarem pacotes e entradas de *logs* e a partir disso gerar um índice de suspeita de ataques [7], [30]. O NIDIA utiliza a metodologia de detecção por abuso e anomalia, garantindo uma robustez maior ao sistema, e também possui a capacidade de interagir com sistemas tipo *firewall* [4], [27].

6.2 Arquitetura Atual do NIDIA

Atualmente, a arquitetura do NIDIA possui 6 (seis) camadas [31] que abrigam entre si agentes que possuem responsabilidades semelhantes. A Figura 6.1 mostra a arquitetura atual do NIDIA.

A camada de monitoramento é responsável por capturar eventos e por fornecer informações sobre os mesmos para o restante do sistema. Esta camada abriga os agentes

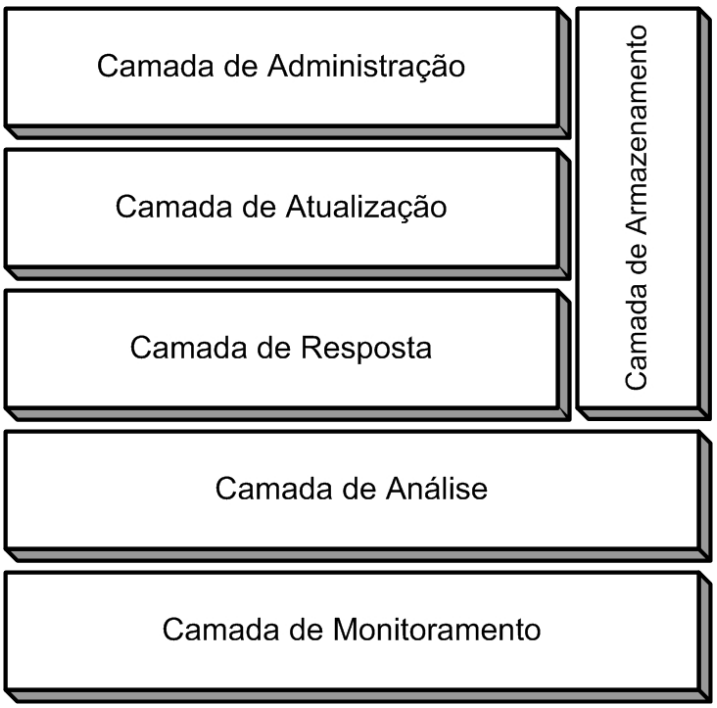


Figura 6.1: Arquitetura Atual do NIDIA

sensores que capturam os pacotes que estão trafegando na rede e que coletam informações em *logs*, em tempo real, de um *host* em particular e disponibilizando-as para a camada de análise.

A camada de análise é responsável pela análise dos eventos recebidos da camada de monitoramento. Nessa camada, os eventos coletados são formatados permitindo que padrões de ataque e/ou comportamentos anormais na rede e nos servidores monitorados sejam identificados. Esta camada também é responsável por emitir um grau de suspeita sobre os eventos que foram previamente formatados utilizando informações específicas da camada de armazenamento. A identificação de ataques também é realizada nesta camada através de um vetor de contagem de palavras chaves e do grau de suspeita de um evento.

A camada de resposta possui vários agentes responsáveis pelo controle das ações que o sistema deve tomar em caso de uma tentativa de invasão. Esta camada recebe as identificações dos ataques, consulta mais informações sobre o ataque e sobre o atacante, avalia o nível de severidade do ataque, consulta as informações da camada de armazenamento e inicia a reação apropriada.

A camada de atualização é responsável pela atualização das bases de dados da camada de armazenamento. As consultas são feitas por agentes de qualquer camada,

porém inserções e atualizações são feitas somente através dos agentes desta camada. Ela possui também a responsabilidade de manter a integridade e consistência das informações armazenadas.

A camada de administração é responsável pela administração e integridade de todos os agentes do sistema. Esta camada busca por eventos inesperados ou que diferem do perfil normal dos agentes ativos. Conseqüentemente, ela interage com todos os agentes do sistema. Nesta camada, também encontra-se a interface de gerenciamento dos agentes utilizada pelo administrador de segurança. Através desta interface é possível gerenciar o estado e a configuração dos agentes, a atualização das bases de dados específicas, o registro das ocorrências detectadas e das ações tomadas pelo sistema como resposta a estas ocorrências.

A camada de armazenamento é responsável por armazenar informações relevantes à detecção de intrusão. Esta camada possui várias bases de dados para diferentes tipos de informações:

- Danos causados por ataques bem sucedidos, tentativas de ataques e ações de resposta que foram tomadas pelo sistema;
- Assinaturas atualizadas de intrusão que serão utilizadas para a detecção de atividades suspeitas;
- Palavras-chaves para definição da severidade de um ataque;
- Informações referentes às ações de resposta que devem ser tomadas de acordo com a severidade de um ataque detectado;
- Estratégias adotadas pela organização em relação a sua política de segurança;
- Listas negras e listas brancas;
- *Logs* de ações executadas nos *hosts*.

6.3 NIDIA como Estudo de Caso

O agente sensor da camada de monitoramento foi utilizado para a aplicação do NIDIA como estudo de caso dos protótipos das soluções de segurança e confiabilidade. O

agente sensor captura pacotes da rede e envia as informações dos pacotes capturados para um agente da camada de análise chamado SMA (Agente de Monitoramento de Sistema).

A Listagem 6.1 representa uma mensagem que representa um pacote capturado pelo agente sensor do NIDIA. O agente sensor utiliza o padrão RDF para que a mensagem utilize a linguagem XML como codificação.

Listagem 6.1: Mensagem de um agente sensor do NIDIA

```
1  <?xml version="1.0"?>
2  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:
       rdfs="http://www.w3.org/TR/1999/PR-rdf-schema-19990303#" xmlns:fipa-
       rdf="http://www.fipa.org/schemas/FIPA-RDF#">
3    <rdf:object>
4      <fipa-rdf:CONTENT_ELEMENT>
5        <rdf:Description>
6          <rdf:type>null#messagem</rdf:type>
7          <IpDestino>192.168.4.138</IpDestino>
8          <Dados>AAAAAABTTUIyAAAAAAAAAAAAAAAAAAAAAAAAAAbAAAAAAAAEwAAAA
               [...]</Dados>
9          <Servico>null</Servico>
10         <Hostorigem>pc01</Hostorigem>
11         <IpOrigem>192.168.4.13</IpOrigem>
12         <NomeAtaque></NomeAtaque>
13         <Resultado></Resultado>
14         <Data>20060317T212246802Z</Data>
15         <PortaDestino>139</PortaDestino>
16         <IndiceSeveridade>0</IndiceSeveridade>
17         <PortaOrigem>1091</PortaOrigem>
18         <Tipo>0</Tipo>
19         <Entrada></Entrada>
20         <HostDestino>ZIM</HostDestino>
21         <Protocolo>TCP</Protocolo>
22       </rdf:Description>
23     </fipa-rdf:CONTENT_ELEMENT>
24   </rdf:object>
25 </rdf:RDF>
```

A Listagem 6.2 mostra o código-fonte do agente sensor. As linhas 1 a 6 ilustram a criação da instância do objeto *security*, a definição do próprio agente sensor, a definição do servidor XKMS e a chamada do método *setup()* da biblioteca. As linha 8 a 11 mostram

a preparação da mensagem a ser enviada ao agente SMA. A utilização do protótipo de confiabilidade é visível na linha 10, onde podemos ver uma atribuição de um endereço de transporte válido para o MTP que representa a solução de confiabilidade. A linha 13 ilustra a chamada ao método *secureMessage* responsável por criptografar e assinar digitalmente a mensagem. A mensagem é enviada na linha 14.

Listagem 6.2: Código-fonte do agente sensor

```
1  Security security = new Security();
2  security.setAgent(this.agent);
3  AID xkmsServer = new AID("xkmsServer@pc01:1099/JADE", AID.ISGUID);
4  xkmsServer.addAddresses("http://192.168.4.130:6060/delivery");
5  security.setXkmsServer(xkmsServer);
6  security.setup();
7  [...]
8  ACLMessage msg = new ACLMessage(ACLMessage.INFORM);
9  AID sma = new AID("sma@pc02:1099/JADE", AID.ISGUID);
10 sma.addAddresses("http://192.168.4.138:6060/delivery");
11 msg.addReceiver(sma);
12 [...]
13 security.secureMessage(msg);
14 agent.send(msg);
```

A Listagem 6.3 mostra o código-fonte do agente SMA. As linhas 1 a 6 ilustram a criação da instância do objeto *security*, a definição do próprio agente SMA, a definição do servidor XKMS e a chamada do método *setup()* da biblioteca. A linha 8 ilustra o recebimento de uma mensagem. A linha 9 mostram a chamada ao método *recoverMessage* que verifica e descriptografa a mensagem recebida.

Listagem 6.3: Código-fonte do agente SMA

```
1  Security security = new Security();
2  security.setAgent(this.agent);
3  AID xkmsServer = new AID("xkmsServer@pc01:1099/JADE", AID.ISGUID);
4  xkmsServer.addAddresses("http://192.168.4.130:6060/delivery");
5  security.setXkmsServer(xkmsServer);
6  security.setup();
7  [...]
8  ACLMessage msg = myAgent.receive();
9  [...]
10 security.recoverMessage(msg);
```

A Listagem 6.4 mostra a mensagem enviada do agente sensor ao agente SMA. O elemento *CipherValue* (linha 13) da mensagem corresponde a mensagem da Listagem 6.1 criptografada. O elemento *CipherValue* (linha 28) corresponde a assinatura digital da mensagem.

Listagem 6.4: Mensagem protegida do agente sensor

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xenc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
       xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
3    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#
         tripledes-cbc"/>
4    <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
5      <xenc:EncryptedKey>
6        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc
             #rsa-1_5"/>
7        <xenc:CipherData>
8          <xenc:CipherValue>B91LxV31iDeSuSU6rHGrlMb4ZTD[...]</xenc:
               CipherValue>
9        </xenc:CipherData>
10     </xenc:EncryptedKey>
11   </ds:KeyInfo>
12   <xenc:CipherData>
13     <xenc:CipherValue>KElORk9STQogOnNlbmRlciAgKCBhZ[...]</xenc:
           CipherValue>
14   </xenc:CipherData>
15   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
16     <ds:SignedInfo>
17       <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/
             REC-xml-c14n-20010315"/>
18       <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
             rsa-sha1"/>
19       <ds:Reference URI="">
20         <ds:Transforms>
21           <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#
                 enveloped-signature"/>
22           <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-
                 c14n-20010315"/>
23         </ds:Transforms>
24         <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#
               sha1"/>
```

```
25          <ds:DigestValue>Oi8vMTkyLjE2OC40LjEzOjYwNjAg</ds:DigestValue>
26        </ds:Reference>
27      </ds:SignedInfo>
28      <ds:SignatureValue>XJkZi1zY2hlbWEtMTk5OTAzMDMj[...]</ds:
            SignatureValue>
29    </ds:Signature>
30  </xenc:EncryptedData>
```

Por utilizar a solução de garantia de entrega de mensagem, o agente sensor recebe uma mensagem de confirmação de entrega do agente SMA. A Listagem 6.5 ilustra a mensagem de confirmação.

Listagem 6.5: Mensagem de confirmação do agente SMA

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <e:Envelope xmlns:d="http://www.w3.org/2001/XMLSchema" xmlns:i="http://
       www.w3.org/2001/XMLSchema-instance" xmlns:wsrm="http://schemas.
       xmlsoap.org/ws/2003/03/rm" xmlns:wsu="http://schemas.xmlsoap.org/ws
       /2002/07/utility" xmlns:e="http://schemas.xmlsoap.org/soap/envelope
       /">
3    <e:Header>
4      <wsrm:SequenceAcknowledgement e:mustUnderstand="1">
5        <wsu:Identifier>urn:6e94c052-a3a3-11da-afbd-6e94c051afbd</wsu:
             Identifier>
6        <wsrm:AcknowledgementRange Upper="1" Lower="1" />
7      </wsrm:SequenceAcknowledgement>
8    </e:Header>
9    <e:Body>
10     <ns0:sendResponse xmlns:ns0="http://systinet.com/wsdl/model/
           reliability/" />
11   </e:Body>
12 </e:Envelope>
```

6.4 Conclusões

O NIDIA mostrou-se um ótimo estudo de caso das soluções de segurança e de confiabilidade, pois mostrou que as soluções de segurança e de garantia de entrega de mensagem realmente possuem uma integração rápida e simples com sistemas multiagentes implementados com o JADE.

A solução de comunicação segura proposta fornece mecanismos de autenticação, confidenciabilidade, não-repúdio do remetente e integridade, como podemos observar na utilização pelo NIDIA.

O mecanismo de garantia de entrega de mensagens fornecido pela solução de confiabilidade também foi integrado com sucesso pelo NIDIA.

7 Conclusão

Este capítulo apresenta as contribuições deste trabalho, as conclusões sobre os resultados alcançados, as limitações das soluções propostas e as sugestões para trabalhos futuros.

7.1 Contribuições

As soluções propostas de segurança e de garantia de entrega de mensagem possibilitam a comunicação segura e confiável aos sistemas multiagentes, dois requisitos importantíssimos para aplicações corporativas, tal como comércio eletrônico, que funcionam na Internet.

A solução de segurança garante mecanismos de confidenciabilidade, integridade, não-repúdio e autenticação para os sistemas multiagentes. A solução de confiabilidade fornece mecanismos de garantia de entrega de mensagens para os sistemas multiagentes.

As duas soluções adaptam especificações XML para obter tais mecanismos fazendo com que as soluções possam interagir com outras aplicações que também utilizem as mesmas especificações. Foi observado que estas especificações XML tinham seu foco na mensagem XML (mensagem SOAP) trocada entre o *web service* e o usuário, então estas especificações poderiam ser adaptadas para sistemas multiagentes, se os agentes "conversassem" utilizando XML. Para tanto, foi utilizado o padrão RDF para que os agentes trocassem entre si mensagens através da linguagem XML.

A confidenciabilidade é obtida através da criptografia fornecida pela especificação XML-Enc (*XML Encrpytion*); a integridade, não-repúdio do remetente e autenticação são obtidas através da assinatura digital fornecida pela especificação XML-DSig (*XML Signature*); A especificação XKMS fornece o suporte ao esquema PKI para o gerenciamento das chaves pública e privada usadas nos procedimentos de criptografia e assinatura digital. Estes mecanismos de segurança fazem parte da solução de segurança. Neste trabalho, um protótipo da solução de segurança é apresentado e foi implementado como uma API Java.

A confiabilidade de comunicação foi obtida através das técnicas de garantia de entrega de mensagem da especificação WS-RM (WS-ReliableMessaging). Neste trabalho, um protótipo da solução de confiabilidade é apresentado e foi implementado como um MTP-JADE.

Como os protótipos das soluções foram implementados utilizando os componentes do *framework* JADE, os protótipos podem ser utilizados por qualquer um que utilize o mesmo *framework*.

Este trabalho também apresentou uma solução de segurança e outra solução de garantia de entrega de mensagens já existente para o JADE. Em seguida, o comparativo entre estas soluções e as soluções propostas foi discutido.

Os protótipos tiveram o IDS NIDIA como estudo de caso.

Este trabalho ainda propôs um modelo de segurança específico ao servidor XKMS que fornece mecanismos de autenticação aos agentes.

Este trabalho ainda ilustrou o desenvolvimento de um MTP-JADE com um pequeno manual prático de como implementar um MTP-JADE.

7.2 Limitações

As soluções de segurança e de confiabilidade possuem algumas limitações:

- A solução de segurança não possui mecanismos de autorização;
- A solução de confiabilidade não possui mecanismos de persistência de envio de mensagens.

Na seção Trabalhos Futuros, apresentamos soluções para estas limitações.

7.3 Considerações Finais

Todos os mecanismos de segurança e de garantia de entrega de mensagens foram obtidos com sucesso nos respectivos protótipos das soluções propostas. Algumas bibliotecas Java foram utilizadas para a adaptação rápida das especializações XML utilizadas.

A solução de segurança teve resultado satisfatório, pois pode ser utilizada em plataformas JADE diferentes.

A solução de confiabilidade proposta possui a vantagem da entrega direta das mensagens ao agentes.

7.4 Trabalhos Futuros

Para a continuidade deste trabalho e sugestão de trabalhos futuros, podemos citar:

- Adicionar ao servidor XKMS uma funcionalidade extra de controlar o tempo de vida de todas as chaves públicas registradas e de alertar os agentes da expiração de tempo de vida das suas chaves públicas;
- Adicionar ao protótipo da solução de segurança a possibilidade de aplicar somente a criptografia ou somente a assinatura digital da mensagem;
- Utilizar os agentes nativos do JADE para realizar tarefas como a solicitação do endereço do servidor XKMS;
- Adicionar mecanismos de autenticação mais robustos e autorização por meios da adaptação de outras especificações XML;
- Adicionar mecanismos de proteção ao arquivo de configuração utilizado na solução de segurança;
- Adicionar mecanismos de *timestamp* às mensagens seguras;
- Adicionar mecanismos de persistência de envio de mensagens à solução de confiabilidade.

Referências Bibliográficas

[1] J. Boyer. Canonical XML Version 1.0 - RFC 3076, 2001.

[2] C. Ferris and D. Langworthy (Eds.). Web Services Reliable Messaging Protocol (WS-ReliableMessaging), 2003. Disponível em http://www-106.ibm.com/developerworks/webservices/library/ws-rm/.

[3] Edward Curry, Desmond Chambers, and Gerard Lyons. A JMS Message Transport Protocol for the JADE Platform. In *IAT '03: Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, page 596, Washington, DC, USA, 2003. IEEE Computer Society.

[4] Glenda de Lourdes Ferreira. Um agente inteligente controlador de ações do sistema. In *Dissertação de Mestrado. Coordenação de Pós-Graduação em Engenharia de Eletricidade. Universidade Federal do Maranhão*, São Luís, Maranhão, Brasil, Novembro 2003.

[5] Stefan Decker, Sergey Melnik, Frank van Harmelen, Dieter Fensel, Michel C. A. Klein, Jeen Broekstra, Michael Erdmann, and Ian Horrocks. The semantic web: The roles of XML and RDF. *IEEE Internet Computing*, 4(5):63 – 74, 2000.

[6] Rômulo Alves Dias. Um modelo de atualização automática do mecanismo de detecção de ataques de rede para sistemas de detecção de intrusão. In *Dissertação de Mestrado. Coordenação de Pós-Graduação em Engenharia de Eletricidade. Universidade Federal do Maranhão*, São Luís, Maranhão, Brasil, Novembro 2003.

[7] Rômulo Alves Dias and Edson Nascimento. Um modelo de atualização automática do mecanismo de detecção de ataques de rede para sistemas de detecção de intrusão. *Proceedings of V Simpósio de Segurança em Informática*, 2003.

[8] D. Dzung, M. Naedele, T. P. Von Hoff, and M. Crevatin. Security for industrial communication systems. *Proceedings of the IEEE*, 93(6):1152 – 1177, Junho 2005.

[9] Donald Eastlake and Joseph Reagle. XML Encryption Syntax and Processing, Dezembro 2002. Disponível em http://www.w3.org/TR/xmlenc-core/.

[10] Donald Eastlake, Joseph Reagle, and David Solo. XML-Signature Syntax and Processing, Fevereiro 2002. Disponível em http://www.w3.org/TR/xmldsig-core/.

[11] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[12] FIPA. FIPA Home Page, 2006. Disponível em http://www.fipa.org.

[13] FIPA. FIPA RDF Content Language Specification, 2006. Disponível em http://www.fipa.org/specs/fipa00011/XC00011B.pdf.

[14] Warwick Ford, Phillip Hallam-Baker, Barbara Fox, Blair Dillaway, Brian LaMacchia, Jeremy Epstein, and Joe Lapp. XML Key Management Specification (XKMS), Março 2001. Disponível em http://www.w3.org/TR/xkms/.

[15] David Geer. Taking steps to secure web services. *IEEE Computer*, 36(10):14 – 16, 2003.

[16] O. Group, Platform, and S. Group. Agent technology – green paper, 2000.

[17] Bret Hartman, Donald J. Flinn, and Shirley Kawamoto. *Mastering Web Services Security*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[18] IETF. IETF Home Page, 2006. Disponível em http://www.ietf.org.

[19] JADE. Class Profile, 2006. Disponível em http://jade.tilab.com/doc/api/jade/core/Profile.html.

[20] JADE. JADE Home Page, 2006. Disponível em http://jade.tilab.com.

[21] Michel Klein. XML, RDF, and Relatives. *IEEE Intelligent Systems*, 16(2):26 – 28, 2001.

[22] O. Lassila and R. Swick. Resource Description Framework (RDF) model and syntax specification.

[23] Christiane Ferreira Lemos Lima. Agentes inteligentes para detecção de intrusos em redes de computadores. In *Dissertação de Mestrado. Coordenação de Pós-Graduação em Engenharia de Eletricidade. Universidade Federal do Maranhão*, São Luís, Maranhão, Brasil, Janeiro 2002.

[24] Microsoft Corporation. Microsoft Home Page, 2006. Disponível em http://www.microsoft.com/.

[25] Haralambos Mouratidis, Paolo Giorgini, and Gordon Manson. Modelling secure multiagent systems. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 859 – 866. ACM Press, 2003.

[26] Martin Naedele. Standards for xml and web services security. *Computer*, 36(4):96 – 98, 2003.

[27] Antonio Alfredo Pires Oliveira, Zair Abdelouahab, and Edson Nascimento. Using honeypots and intelligent agents in security incident responses and investigation of suspicious actions in interconnected computer systems. In *Proceedings of the E-Crime And Computer Evidence Conference Mônaco*, Monaco, 2005.

[28] Emerson Oliveira, Zair Abdelouahab, and Denivaldo Lopes. Security on MASs with XML Security Specifications. In *DEXA '06: Proceedings of the 17th International Conference on Database and Expert Systems Applications*, pages 5–9, Washington, DC, USA, 2006. IEEE Computer Society.

[29] S. Poslad and M. Calisti. Towards improved trust and security in fipa agent platforms. In *Proceedings of Autonomous Agents 2000 Workshop on Deception, Fraud and Trust in Agent Societies*, Junho 1999.

[30] Glenda Lourdes Ferreira Santos and Edson Nascimento. An automated response approach for intrusion detection security enhancement. *Proceedings of VII International Conference on Software Engineering and Applications*, 2003.

[31] Lindonete Gonçalves Siqueira. Tolerância a falhas no network intrusion detection system based on intelligent agents (nidia). In *Dissertação de Mestrado. Coordenação de Pós-Graduação em Engenharia de Eletricidade. Universidade Federal do Maranhão*, São Luís, Maranhão, Brasil, 2006.

[32] SUN. Class URL, 2006. Disponível em http://java.sun.com/j2se/1.4.2/docs/api/java/net/URL.html.

[33] SUN. Java Architecture for XML Binding (JAXB), 2006. Disponível em http://java.sun.com/webservices/jaxb/index.jsp.

[34] Sun Microsystems. Java Authentication and Authorization Service (JAAS) 1.0 Developer's Guide, 2006. Disponível em http://java.sun.com/products/jaas.

[35] Sun Microsystems. Java Message Service Specification, 2006. Disponível em http://java.sun.com/products/jms.

[36] Systinet. Systinet Home Page, 2006. Disponível em http://http://www.systinet.com.

[37] Systinet. Systinet Server for Java 6.5.3 API, 2006. Disponível em http://www.systinet.com/doc/ssj-60/api/index.html.

[38] Andrew Tanenbaum. *Redes de Computadores*. Editora Campus, 4ª edition, 2003.

[39] VeriSign. VeriSign Home Page, 2006. Disponível em http://www.verisign.com.

[40] W3C. W3C Home Page, 2006. Disponível em http://www.w3.org.

[41] WebMethods Inc. WebMethods Home Page, 2006. Disponível em http://webmethods.com/.

[42] P. Wing and B. O'Higgins. Using public-key infrastructure for security and risk management. *IEEE Communications Magazine*, 37(9):71 – 73, Setembro 1999.

[43] Hao Chi Wong and Katia Sycara. Adding security and trust to multi-agent systems. In *Proceedings of Autonomous Agents '99 Workshop on Deception, Fraud, and Trust in Agent Societies*, pages 149 – 161, Maio 1999.