André Detsch

Uma Arquitetura para Incorporação Modular de Aspectos de Segurança em Aplicações Peer-to-Peer

Dissertação submetida a avaliação como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada

Orientador: Prof Dr. Luciano Paschoal Gaspary

Co-orientador: Prof Dr. Marinho Pilla Barcellos

São Leopoldo
2005

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Detsch, André

Uma Arquitetura para Incorporação Modular de Aspectos de Segurança em Aplicações Peer-to-Peer / por André Detsch. — São Leopoldo: Ciências Exatas e Tecnologicas da UNISINOS, 2005.

95 f.: il.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos. Ciências Exatas e Tecnológicas Curso de Pós-Graduação em Computação Aplicada, São Leopoldo, BR–RS, 2005. Orientador: Gaspary, Luciano Paschoal; Co-orientador: Barcellos, Marinho Pilla.

1. Peer-to-Peer. 2. Segurança. 3. JXTA. I. Gaspary, Luciano Paschoal. II. Barcellos, Marinho Pilla. III. Título.

Sumário

Lista de Figuras

Lista de Abreviaturas

ACL	Access Control List
CA	Certification Authority
CMS	Cryptographic Message Syntax
DES	Data Encryption Standart
DSA	Digital Signature Algorithm
DTD	Document Type Definitions
GRAM	Globus Resource Allocation Manager
GSI	Grid Security Infrastructure
HTTP	Hiper-Text Tranport Protocol
IP	Internet Protocol
JAL	JXTA Abstraction Layer
JXTA	Juxtapose
MAC	Message Authentication Code
MD5	Message-Digest Algorithm
MDS	Monitoring and Discovery Service
NAT	Network Address Translation
OGSA	Open Grid Service Architecture
P2P	Peer-to-Peer
PGP	Pretty Good Privacy

RC4	Rivest Cipher 4
RSA	Rivest, Shamir and Adelman
RTF	Reliable Transfer Infrastructure
SHA1	Secure Hash Algorithm - Version 1.0
SOAP	Simple Object Access Protocol
SSL	Session Layer Security
TCP	Transport Control Protocol
TDEA	Triple Data Encryption Algorithm
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
WSDL	Web Service Definition Language
WSS	Web Service Security
XML	Extensible Markup Language

Resumo

Aplicações Peer-to-Peer (P2P) vêm ganhando espaço em ambientes acadêmicos e, sobretudo, corporativos. Nesses ambientes, a ampla adoção de tais aplicações está condicionada ao atendimento de diversos requisitos de segurança, como autenticação, confidencialidade e autorização. Existem algumas soluções que buscam suprir necessidades isoladas de segurança, mas possuem a desvantagem de não poderem ser usadas de maneira integrada e exigirem do usuário e do programador das aplicações a manipulação de uma interface de programação complexa, seguida de uma configuração exaustiva. Esta dissertação apresenta uma arquitetura de segurança que isola da aplicação P2P e do middleware de comunicação subjacente a implementação e a configuração dos aspectos de segurança. Esses são atendidos através de módulos isolados que implementam técnicas de segurança já consolidadas. A arquitetura foi implementada tendo como base JXTA, um middleware P2P bastante disseminado. Como estudo de caso, a arquitetura foi incorporada ao OurGrid, aplicação de grade computacional baseada no paradigma peer-to-peer.

Palavras-chave: Peer-to-Peer, Segurança, JXTA.

TITLE: "An Architecture for the Modular Inclusion of Security Aspects in Peer-to-Peer Applications"

Abstract

Peer-to-peer (P2P) applications are becoming increasingly important in academic environments and, in particular, corporative ones. In such environments, the wide adoption of P2P applications is subject to the fulfilment of several security requirements, including authentication, confidentiality, and authorization. There exist approaches that aim to provide specific security needs, but have two disadvantages: cannot be employed integratedly, and require the user/application programmer to manipulate a complex programming interface, followed by a cumbersome configuring process. This thesis presents a security architecture that isolates from the P2P application and the underlying communication middleware the implementation and configuration of security aspects. The latter are achieved through independent modules that implement well-known security techniques. The architecture was implemented using JXTA, a popular P2P middleware. As a study case, the architecture was incorporated to OurGrid, an application of Grid Computing that is based on the peer-to-peer paradigm.

Keywords: Peer-to-Peer, Security, JXTA.

Capítulo 1

Introdução

1.1 Contextualização

Redes Peer-to-Peer (P2P) apareceram recentemente como um novo paradigma para construção de aplicações distribuídas. A abordagem P2P difere em vários aspectos do modelo cliente-servidor, sendo a principal diferença o fato de os *peers* atuarem tanto como cliente quando servidores dos serviços envolvidos na aplicação ([1]).

Apesar de redes P2P terem sido foco de uma grande variedade de recentes projetos de pesquisa, muitas questões ainda estão em aberto, onde se sobressaem os aspectos relativos a aplicações com requisitos de segurança. Quando se trata de segurança em redes P2P, pode-se identificar dois grandes campos de pesquisa.

Um deles se refere a garantir a segurança de uma rede / instituição quando do uso de aplicações P2P quaisquer por parte de seus integrantes ([2]). Fazem parte desse escopo medidas como bloquear o tráfego gerado por aplicações de compartilhamento de arquivos (para que a banda da rede não fique comprometida) ou tentar evitar que tais aplicações sirvam de porta de entrada para vírus ou *trojan horses*.

O segundo campo de pesquisa tem por objetivo possibilitar a criação de aplicações P2P seguras no que diz respeito ao seu funcionamento interno ([3, 4]). Isto implica possibilitar ao programador de aplicações P2P a adição de diferentes aspectos de segurança, como autenticação, confidencialidade, integridade e autorização. É nesta área que a dissertação está focada, ou seja, na construção de sistemas P2P seguros.

1.2 Definição do problema

Com a expansão do interesse de uso de aplicações P2P, especialmente em ambientes corporativos, a segurança desse tipo de sistema acaba sendo um dos principais entraves para a sua franca utilização ([5]). A implementação de aspectos de segurança se faz necessária, mas, ao mesmo tempo, pode dificultar muito a criação de um sistema.

Propostas atuais que visam atender a essa questão oferecem soluções específicas, que não tratam os diversos aspectos de segurança dentro de um mesmo modelo. Em geral, os estudos nesta área abordam apenas algum aspecto em especial, como autorização ([6]) ou autenticação ([7]). Ferramentas de implementação, como *toolkits* ou bibliotecas, apesar de atenderem a mais de um aspecto de segurança, não estabelecem um *framework* para sua utilização ([4]).

Outra limitação das soluções disponíveis é a necessidade de implementação dos aspectos de segurança de forma embutida na aplicação, aumentando a complexidade do processo de criação e manutenção do código. Essa abordagem traz dificuldades ainda maiores se considerarmos a variabilidade típica em redes P2P. Aspectos de segurança podem passar a ser necessários ou desnecessários de forma bastante dinâmica, exigindo que a parte da aplicação referente ao tratamento desses aspectos tenha que ser constantemente ajustada e reconfigurada.

Considerando especialmente a utilização de sistemas P2P que abrangem diversas instituições, surge a necessidade de diferenciar os aspectos necessários para cada grupo de nodos presente na rede. A comunicação com nodos de uma rede local, por exemplo, pode ter requisitos de segurança menos rígidos, ao mesmo tempo em que a utilização mais avançada de aspectos de segurança continua importante para nodos remotos. Aliado a essa necessidade, é importante permitir a comunicação mesmo com nodos que não implementam os aspectos de segurança, sem exigir que a atualização em um sistema P2P tenha que ser feita de forma simultânea em todas as estações.

Por fim, é um problema comum em aplicações P2P a não adequação a tecnologias de segurança comumente empregadas, em especial *firewalls*. Essa característica acaba por elevar o custo de implantação de um novo sistema, pois acarreta a necessidade de reconfiguração de dispositivos em função das características da aplicação.

1.3 Objetivos

Este trabalho tem por objetivo especificar, implementar e validar uma arquitetura para incorporação de aspectos de segurança em aplicações P2P que permita:

- o isolamento da codificação e da configuração de aspectos de segurança do restante da aplicação P2P;
- a selecionação individual dos aspectos de segurança a serem atendidos;
- a comunicação com entidades do mesmo sistema que não implementem os aspectos de segurança;
- a co-existência com as estruturas de segurança utilizadas atualmente (baseadas, por exemplo, em *firewalls*).

A arquitetura proposta se baseia na criação de uma camada de segurança que abstrai (da aplicação) tanto a implementação dos aspectos de segurança quanto a configuração dos mesmos. A arquitetura foi implementada sobre JXTA ([8]), um conjunto de protocolos que permite a criação de uma vasta gama de aplicações P2P. A validação foi realizada através do uso da arquitetura em conjunto com o OurGrid ([9]), uma aplicação P2P para o estabelecimento de grades computacionais. Esta etapa permitiu avaliar tanto a adequação da solução a um caso real quanto identificar características de desempenho da mesma.

1.4 Organização da dissertação

O restante da dissertação está organizado conforme listado a seguir. O Capítulo 2 apresenta conceitos relevantes no decorrer da dissertação, em especial no que diz respeito a redes P2P e segurança computacional. O Capítulo 3 introduz trabalhos relacionados que constituem alternativas para a construção de aplicações P2P incorporando aspectos de segurança. O Capítulo 4 apresenta a arquitetura proposta, descrevendo os seus principais elementos. O Capítulo 5 descreve a implementação realizada, seguida das análises de uso e sobrecarga realizadas tendo como base o OurGrid (Capítulo 6). O Capítulo 7 encerra a dissertação com considerações finais e perspectivas de trabalhos futuros.

Capítulo 2

Fundamentos de segurança e redes Peer-to-Peer

Este capítulo apresenta uma fundamentação sobre redes Peer-to-Peer (Seção 2.1) e aspectos de segurança importantes em redes de computadores (Seção 2.2), ambos temas relevantes no decorrer do texto. A Seção 2.3 finaliza o capítulo, descrevendo como os aspectos de segurança são necessários em cada uma das diferentes categorias de aplicações P2P.

2.1 Redes Peer-to-Peer

Não existe um consenso na definição exata do que seja uma rede Peer-to-Peer. A definição depende, em geral, do contexto em que a tecnologia é empregada. De forma geral, entretanto, estabelece-se que redes P2P são redes virtuais que funcionam na Internet com o objetivo de compartilhar recursos entre os participantes, sendo que, por princípio, não há diferenciação entre os participantes ([10]). O grupo de pesquisa sobre redes P2P da IRTF [11] as define como o "compartilhamento de recursos e serviços computacionais diretamente entre sistemas". Aplicações ou sistemas, por sua vez, são classificados como P2P quando sua lógica de funcionamento se baseia no paradigma de redes P2P. Em geral, é aceito pela comunidade que sistemas P2P devem suportar os seguintes requisitos ([10]):

- possibilidade de incorporação de nós localizados nas bordas da rede;
- suporte à conectividade variável ou temporária dos nós, bem como a utilização de endereços variáveis;

- capacidade de lidar com diferentes taxas de transmissão entre os nós;
- autonomia parcial ou total dos nós em relação a um servidor centralizado;
- escalabilidade;
- possibilidade de comunicação direta entre os nós.

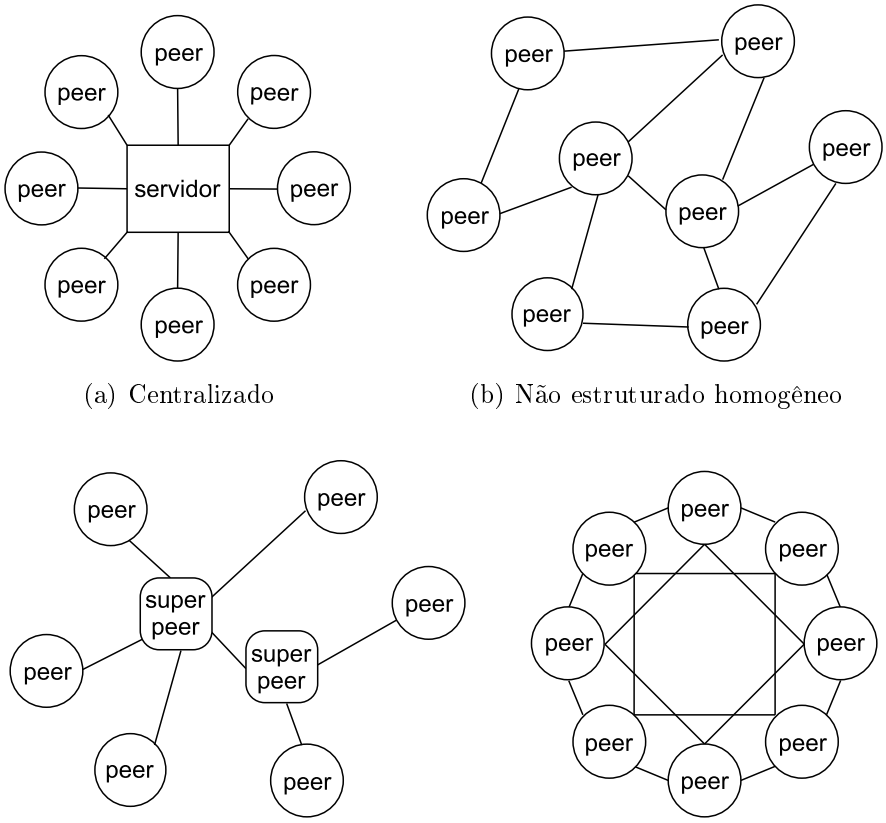
Tendo todas essas características, uma rede pode ser dita Peer-to-Peer, mesmo que algumas das funções de controle da rede estejam localizadas em um servidor central (ponto de falha).

O modelo P2P é atrativo por uma série de razões, conforme enumerado a seguir. Primeiro, sistemas P2P provêem mecanismos para agregar recursos distribuídos geograficamente em um grupo, sem que todos os membros precisem estar conectados simultaneamente. Segundo, o custo de criar um ambiente colaborativo é comparativamente baixo, uma vez que nenhuma estrutura adicional de hardware se faz necessária. Terceiro, sistemas P2P são inerentemente mais tolerantes a falhas que os baseados no modelo cliente-servidor, uma vez que não existe um ponto central de falhas, sendo mais resistentes a ataques intencionais (por exemplo, de *Denial-of-Service*). Por fim, sistemas P2P se adaptam bem tanto à pequena quanto à grande escala. Sua escalabilidade decorre do fato de que ao mesmo tempo em que a existência de um número maior de nodos acarreta uma maior demanda por recursos, existe uma tendência natural de aumentar também o número de nodos que oferecem os recursos. Adicionalmente, esquemas de replicação ([12]) contribuem para distribuir a carga entre os nodos que compõem a rede.

Em contrapartida, o desenvolvimento de sistemas P2P deve levar em conta algumas características que advém da dinamicidade deste modelo. Em especial, os *peers* podem entrar e sair da rede com grande freqüência e são tipicamente bastante heterogêneos. Buscando lidar com estas características, existem diferentes modelos de redes P2P comumente empregados, descritos a seguir.

Modelos de Redes P2P

Redes P2P podem ser classificadas em três arquiteturas ([13]): centralizada, descentralizada não-estruturada e descentralizada estruturada. O modelo centralizado (Figura 2.1 (a)), empregado por aplicações com o Napster, SETI@Home e ICQ, lança mão de uma entidade central para executar uma função chave no sistema. No caso do Napster, por exemplo, os *peers*, ao ingressarem na rede, transmitem a informação dos arquivos que possuem a um servidor central, que indexa as informações recebidas. O processo



(a) Centralizado (b) Não estruturado homogêneo

(c) Não estruturado heterogêneo (d) Estruturado

FIGURA 2.1 – Modelos de redes P2P.

de busca por arquivos se dá consultando diretamente o servidor central, que repassa a informação de quais *peers* possuem os dados sendo procurados para que, então, o arquivo seja transferido através de uma comunicação P2P.

No modelo descentralizado não-estruturado, forma-se uma malha de comunicação entre os *peers* sem que haja uma unidade central para indexação. Na sua forma mais simples (Figura 2.1 (b)), todos os nodos são responsáveis por rotear mensagens como as de busca e os recursos são indexados apenas no *peer* responsável pelos mesmo. Isto implica na necessidade de um algoritmo de inundação para que algum recurso seja encontrado na rede, já que não há como saber *a priori* onde algum dado pode ser encontrado com maior probabilidade. Uma evolução sobre este modelo básico (Figura 2.1 (c)) é o aproveitamento da heterogeneidade tipicamente existente nos nodos que compõem a rede. Nós com maior conectividade e maior poder de processamento podem acumular tarefas chave no sistema. Por exemplo, a indexação dos dados pode ser realizada completamente em um conjunto de

nodos centrais, também chamados de *superpeers*, reduzindo o custo resultante do processo de busca por recursos. Aos nodos folha cabe apenas indexar os seus dados nos nodos centrais e atender as requisições sobre os seus recursos locais. O modelo descentralizado não-estruturado é empregado por aplicações como o Kazaa ([14]) e o Gnutella ([15]).

Já no modelo descentralizado estruturado (Figura 2.1 (d)), estabelece-se uma estrutura de indexação global, que direciona mensagens de busca para os *peers* que possuam os dados ou recursos sendo procurados. Forma-se uma estrutura organizada de ligação entre os nós, visando alcançar uma boa relação entre as ligações entre os nós e o número de *hops* médio necessário para rotear uma mensagem de busca ([16]). Neste contexto, se destacam sistemas que fazem uso de DHT (*Distributed Hash Table*), como Chord ([17]) ou CAN ([18]), onde os recursos (sejam eles dados, serviços ou mesmo *peers*) são indexados através de chaves únicas no sistema. A indexação dos recursos é realizada tendo como base alguma estrutura lógica que permita o agrupamento de chaves similares, permitindo um roteamento direcionado para os *peers* que contenham índices para o conteúdo desejado.

Exemplos de aplicações

Apesar de serem os exemplos mais clássicos, aplicações P2P não estão restritas a grandes sistemas de compartilhamento de dados ou de troca de mensagens. São cada vez mais comuns aplicações de menor escala, onde um grupo da ordem de dezenas ou centenas de nodos estabelecem um sistema específico, como compartilhamento de recursos computacionais ([9]) ou trabalho cooperativo ([19]). Também no contexto empresarial P2P ganha espaço através da construção de sistemas, seja para integração ou troca de serviços entre instituições, seja para uso interno entre setores ou funcionários ([5]). De forma geral, pode-se identificar várias aplicações para redes P2P.

- Troca de mensagens (*instant messaging*): permite a troca de mensagens de texto entre usuários da rede. Diferentemente do correio eletrônico, em que uma mensagem é armazenada em uma caixa postal e posteriormente entregue ao usuário que verificou a caixa postal no seu servidor, os sistemas de troca de mensagens possibilitam a entrega imediata ao usuário. São exemplos de aplicações deste tipo ICQ ([20]) e Jabber ([21]).

- Distribuição de conteúdo: Dados, em geral na forma de arquivos, são compartilhados entre os usuários da rede P2P. Aplicações deste tipo envolvem a indexação dos dados disponibilizados, operações de busca e conexões diretas entre os *peers* para

a transferência dos dados. Diferentes aplicações que se enquadram nesta categoria possuem diferentes características e mesmo diferentes objetivos de uso. Aplicações como Napster e Gnutella (que seguem, respectivamente, os modelos centralizado e descentralizado não-estruturado) têm por objetivo simplesmente o compartilhamento de arquivos entre usuários. O Publius ([22]) e a Freenet ([23]) buscam também garantir a anonimicidade dos autores e consumidores dos dados. Já o Chord (modelo descentralizado estruturado) tem por principal objetivo garantir a alta disponibilidade dos dados, através de uma indexação eficiente (usando DHT) e da replicação dos dados em diferentes nós.

- Grades computacionais: recursos computacionais distribuídos, em especial, os recursos ociosos, são compartilhados entre os integrantes da grade. Este tipo de aplicação envolve a localização de recursos adequados, transferência de tarefas e de resultados. Dois exemplos de aplicações de grade computacional que usam uma abordagem P2P são SETI@home (modelo centralizado) e OurGrid (descentralizado), descrito em mais detalhes na Seção 6.1.

- Trabalho colaborativo - Os programas de trabalho colaborativo ou *groupware* são projetados para melhorar a produtividade de indivíduos com metas e interesses comuns ([10]). A expressão *groupware* é definida como um *software* que suporta colaboração, comunicação e coordenação de vários usuários em uma rede. Isto inclui a integração de funcionalidades como e-mail, calendário, espaços de trabalho, listas de discussão, sistemas de gerenciamento de documentos, vídeo conferência, entre outras. A abordagem P2P permite a esse tipo de aplicação integrar os diversos participantes de um sistema sem a necessidade de uma estrutura de rede especial. A seguir são citadas algumas das principais aplicações de trabalho colaborativo baseadas em redes P2P. O Groove ([24]) usa espaços de trabalho que são criados e compartilhados entre os participantes. Ele usa um mecanismo centralizado para encontrar os *peers* e iniciar os serviços. Após esta etapa, toda a comunicação acontece diretamente entre os *peers* participantes. O Consilient ([25]) fornece um arcabouço ([10]) para o desenvolvimento e o gerenciamento do fluxo de processos de uma empresa. Aplicações de videoconferência, como Skipe ([26]) e NetMeeting ([27]), também se encaixam nesta categoria.

- Jogos on-line: redes P2P são usadas como base para as inúmeras comunidades de jogos on-line ([10]). Em especial, o uso de uma estrutura P2P se torna uma esco-

lha natural para jogos massivamente paralelos ([28]), onde milhares de jogadores co-existem em um mesmo mundo virtual. Entretanto, enquanto abordagens centralizadas são comuns, estruturas P2P descentralizadas para jogos em grande escala ainda se restringem a estudos científicos ([28]).

Um dos principais campos de estudo em redes P2P é a incorporação de funcionalidades de segurança às aplicações ([29]). A seção a seguir apresenta os principais aspectos de segurança empregados em aplicações de redes de computadores. A relevância de cada um deles em uma rede P2P varia conforme o tipo de aplicação, conforme será discutido na Seção 2.3.

2.2 Fundamentos de segurança

A segurança em redes de computadores abrange diversos aspectos, que atendem a diferentes objetivos do ponto de vista dos usuários das aplicações. Esta seção descreve os principais aspectos de segurança relevantes neste contexto, apresentando os seus elementos e as técnicas empregadas com o objetivo de atendê-los.

2.2.1 Confidencialidade

Confidencialidade é o aspecto que garante a proteção dos dados transmitidos quanto a sua monitoração por parte de entidades não autorizadas. A confidencialidade pode ser aplicada em vários níveis, desde toda uma transmissão de dados entre dois usuários até individualmente por mensagem ou mesmo protegendo apenas campos específicos de uma mensagem. Adicionalmente, confidencialidade pode também estar relacionada a evitar a análise do tráfego por terceiros através da identificação de origem/destino, freqüência, duração ou outras características do fluxo de mensagens.

A garantia de confidencialidade dos dados está relacionada ao uso de um algoritmo de cifragem, ou criptografia. Algoritmos com esta finalidade podem ser divididos em dois grupos: criptografia simétrica e criptografia assimétrica ([30]).

Em algoritmos de *criptografia simétrica*, ambas as partes comunicantes compartilham uma única chave secreta que é usada tanto na cifragem (no remetente) quanto na decodificação (no destino). Os algoritmos mais difundidos nesta categoria são o *Data Encryption Standard* (DES), o *Triple Data Encryption Algorithm* (TDEA), também referenciado como *triple-DES,* e o *Rivest Cipher 4* (RC4).

O *DES* divide a mensagem original em blocos de 64 bits. A execução de uma série de operações sobre cada bloco utilizando como base uma chave de 56 bits resulta em um bloco cifrado, também com 64 bits. A descrição detalhada do algoritmo pode ser encontrada em [31]. No lado do receptor, as operações são realizadas na ordem inversa, utilizando a mesma chave, recuperando-se a mensagem original. Apesar de sua popularidade, principalmente entre 1977 e início dos anos 90, seu uso atualmente não é indicado para dados críticos, já que a evolução na velocidade dos computadores fez com que mensagens cifradas com DES possam ser decodificadas em poucas horas utilizando força bruta ([30]). Esta limitação se deve ao reduzido tamanho da chave (56 bits).

O *TDEA* se baseia no próprio DES para obter uma cifragem mais segura: são utilizadas três chaves de 56 bits e três execuções consecutivas do DES. Isto resulta em uma chave efetiva de 168 bits, virtualmente impossível de ser quebrada usando força bruta ([30]).

O *RC4*, ao contrário do DES e do TDEA, é orientado a fluxo (*stream*), e não a bloco de dados, ou seja, ao invés de quebrar a mensagem em blocos de tamanho fixo (64 bits, por exemplo), a cifragem é feita bit a bit. RC4 pode ser utilizado com uma chave de tamanho arbitrário, entre 1 e 2048 bits. A execução do algoritmo RC4 é da ordem de 10 vezes mais rápida que o DES ([32]).

No caso da *criptografia assimétrica*, o processo de comunicação segura envolve o uso de duas chaves distintas: uma para cifragem e outra para decodificação. A chave para cifragem pode ser distribuída livremente, sendo por isso chamada também de chave pública. Um dos algoritmos de criptografia assimétrica mais difundidos é o *RSA* ([33]). A exemplo do DES e do TDEA, seu funcionamento é orientado a blocos de dados, com a diferença de que é permitido o uso de um tamanho arbitrário de bloco. O algoritmo possibilita também o uso de chaves de qualquer tamanho, sendo 768, 1024 e 2048 bits tamanhos típicos.

Troca de chaves

Uma das principais etapas da comunicação utilizando cifragem está na troca de chaves entre as partes. Para o caso de criptografia simétrica, a princípio é necessário o uso de um canal seguro para transmissão da chave, ou seja, o emprego de alguma conexão criptografada com outra chave (usando uma terceira entidade como intermediária, por exemplo). Pode-se também realizar a troca de chaves por um meio físico, sem depender da rede. Por fim, outra forma é utilizar algum protocolo de troca de chaves, como o

Diffie-Hellman ([34]). Este protocolo se baseia em chaves assimétricas para permitir a troca de uma chave simétrica. Apesar de dispensar a existência de um canal seguro entre as entidades, o algoritmo de Diffie-Hellman, na sua implementação mais simples, não garante a autenticidade das partes, fazendo-se necessário um mecanismo de autenticação.

Já na criptografia assimétrica, a troca de chaves pode ser feita de maneira mais simples, posto que a chave pública de uma entidade pode ser transmitida livremente, sem a necessidade de um canal seguro entre as estações. Entretanto, a simples transmissão de uma chave pública entre duas entidades A e B não garante a autenticidade das partes: uma terceira entidade C pode se fazer passar por A enviando uma chave pública falsa. Para evitar esse tipo de falsificação, podem ser usados *certificados digitais*, que utilizam a assinatura de uma entidade confiável (*Certification Authority*) para garantir a autenticidade nas transmissões.

Outra possibilidade para assegurar a validade das chaves públicas é o uso de *Pretty Good Privacy* (PGP - [7, 35]), onde as chaves públicas são distribuídas entre os nodos sem a necessidade de certificados digitais. Cada nodo determina quais chaves ele sabe que são válidas, assinando-as com a própria chave. Chaves públicas assinadas por entidades conhecidas também são consideradas válidas. Seguindo esta lógica, PGP cria uma teia de confiança entre os integrantes da rede, possibilitando a verificação de autenticidade de mensagens mesmo sem uma entidade central confiável.

2.2.2 Autenticação

A autenticação tem por objetivo assegurar que o remetente da mensagem é de fato quem afirma ser. Em uma transação ou em uma conexão mantida entre duas estações, dois aspectos estão envolvidos. Primeiro, no estabelecimento da conexão, o serviço assegura que ambas as entidades são autênticas. Segundo, o serviço assegura que a conexão não sofre interferência de forma que uma terceira entidade possa se fazer passar por uma das partes com o propósito de envio ou recebimento não autorizado de mensagens.

Autenticação com criptografia

É possível prover autenticação simplesmente usando *criptografia simétrica*. Assumindo que apenas o remetente e o receptor compartilham uma chave, então apenas o transmissor genuíno é capaz de cifrar corretamente a mensagem para o outro participante.

De forma similar, o transmissor da mensagem pode usar um algoritmo de *cripto-*

grafia assimétrica, mas, ao invés de realizar a etapa de cifragem com a chave pública (do receptor), ele utiliza a sua própria chave privada. O receptor, ao conseguir decodificar a mensagem utilizando a chave pública referente ao transmissor, saberá que o remetente realmente é quem afirma ser, já que apenas o transmissor conhece a chave privada correspondente à chave pública utilizada na decodificação. A utilização desta técnica isoladamente não garante a confidencialidade dos dados transmitidos, uma vez que a chave relativa à decodificação da mensagem é pública.

Autenticação sem criptografia

Apesar de ser possível encriptar e prover autenticação em uma só etapa, muitas vezes é interessante permitir a autenticação sem a necessidade de cifrar a mensagem em si, utilizando apenas uma *tag* de autenticação. Isto permite que a mensagem possa ser lida normalmente, sem necessidade de passar por um processo prévio de decodificação. A verificação da assinatura pode ser feita apenas quando necessário (ou através de amostragem sobre as mensagens recebidas), reduzindo o custo total de processamento. A seguir são apresentadas as duas principais técnicas que utilizam esta abordagem.

A técnica *código de autenticação de mensagem* (*Message Authentication Code* - MAC) envolve a utilização de uma chave secreta, compartilhada apenas entre as duas partes comunicantes, para geração de um pequeno bloco de dados, o código de autenticação. A Figura 2.2 ilustra o funcionamento do mecanismo. Quando uma mensagem é transmitida, realiza-se também o envio do código de autenticação, que é calculado em função da mensagem e da chave secreta. O receptor calcula para a mensagem recebida (usando a chave secreta) o código de autenticação resultante. Se este código coincidir com o código recebido, a mensagem é autêntica. O algoritmo para obtenção do código pode ser o próprio DES: gera-se a versão cifrada da mensagem e utiliza-se os últimos bits resultantes (tipicamente 16 ou 32 bits) como código de autenticação.

Uma técnica similar ao código de autenticação de mensagem é o uso de uma *função hash não-reversível*, como MD5 ([36]) ou SHA1 ([37]). A exemplo do código de autenticação de mensagem, uma função *hash* aceita uma mensagem de tamanho arbitrário e gera um verificador (*digest*) de tamanho fixo. No entanto, nenhuma chave secreta é utilizada nesta fase do processo. Através do uso de funções *hash* não-reversíveis, existem três formas de garantir a autenticidade das mensagens, ilustradas na Figura 2.3:

- Criptografia simétrica (a): o verificador gerado é cifrado e enviado juntamente com a mensagem. O receptor realiza a decodificação deste verificador e compara com o

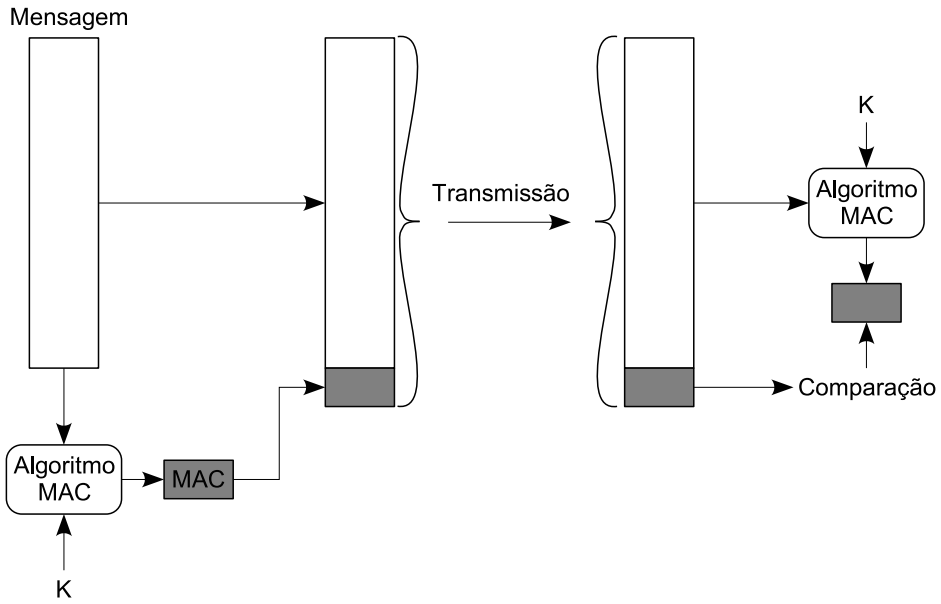


FIGURA 2.2 – Autenticação de Mensagem utilizando código de autenticação (MAC).

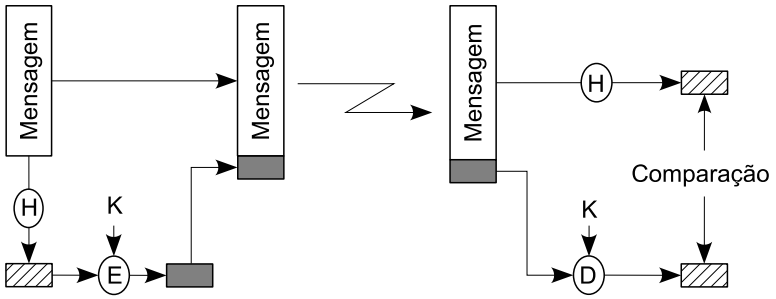
verificador resultante da aplicação da função *hash* sobre a mensagem recebida. Uma terceira entidade que tente forjar uma mensagem consegue gerar o verificador, mas não terá a chave da qual depende a cifragem correta do mesmo.

- Criptografia assimétrica (b): o verificador da mensagem original é cifrado com a chave privada do transmissor, e conferido no recebimento através da decodificação com a respectiva chave pública;

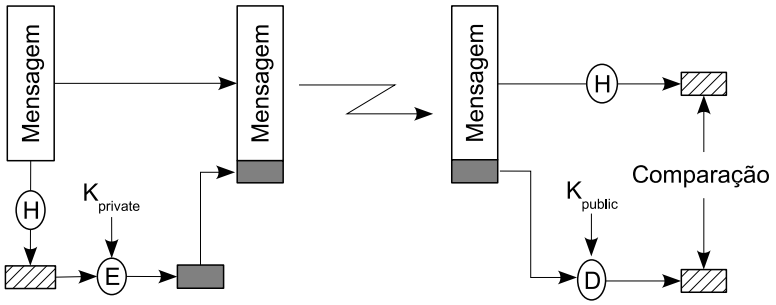
- Valor secreto (c): um bloco secreto é compartilhado apenas entre as entidades comunicantes. Tanto no envio quanto no recebimento de uma mensagem, este bloco é concatenado aos dados apenas para a realização do cálculo do verificador. Esta técnica utiliza o fato de que o resultado da aplicação de uma função *hash* segura depende de todos os bits da entrada, ou seja, não é possível gerar um verificador correto sem que se conheça os bits (no caso, mensagem + bloco secreto).

2.2.3 Integridade

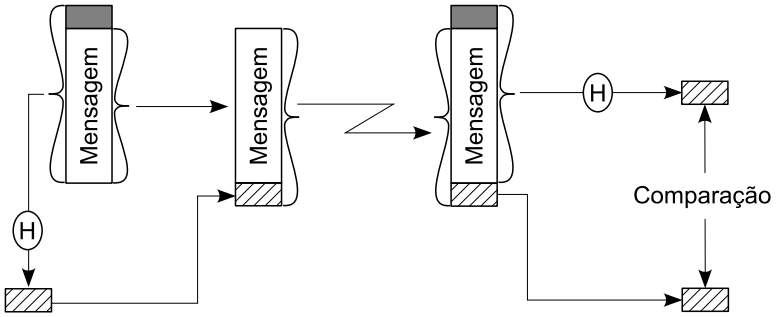
Assim como confidencialidade, a integridade pode ser aplicada a um fluxo de mensagens, a uma mensagem individual ou a alguns campos da mensagem. Um serviço de



(a) Utilizando criptografia convencional



(b) Utilizando criptografia assimétrica



(c) Utilizando valor secreto

FIGURA 2.3 – Autenticação de mensagem utilizando função *hash* não-reversível.

integridade orientado à conexão (aplicado a um fluxo de mensagens) garante que as mensagens são recebidas da mesma forma como enviadas, sem duplicação, inserção, modificação ou reordenamento. Já um serviço de integridade baseado em mensagens individuais provê apenas garantias quanto à modificação de mensagens.

Serviços de integridade podem ser diferenciados também pela existência ou não de recuperação (*recovery*). Quando uma violação de integridade é detectada, o serviço pode simplesmente reportar esta violação ou prover mecanismos para automatizar a recuperação dos dados alterados, sem necessidade de intervenção externa.

Do ponto de vista das técnicas empregadas, a verificação de integridade se assemelha bastante à autenticação. Pode ser utilizada criptografia simétrica ou assimétrica, desde que à mensagem seja adicionado um verificador (que pode ser gerado através de uma função *hash*, reversível ou não). O uso de código de autenticação de mensagem ou de uma técnica que envolve o uso de uma função *hash* segura também garante a integridade dos dados, pois em ambos os casos, um eventual ataque não terá como alterar a mensagem e, ao mesmo tempo, o código ou verificador de forma adequada.

2.2.4 Não-repúdio

Não-repúdio previne que alguma das entidades envolvidas em uma comunicação possa negar alguma operação, como envio ou recebimento de determinada mensagem. Sua principal aplicação está em transações comerciais ou financeiras, onde, eventualmente, uma das partes poderia levar vantagem ao negar a realização de alguma transação.

As técnicas de não-repúdio envolvem a utilização de mecanismos de autenticação e integridade, além de uma entidade (TTP - *Trusted Third Party*) que registra os eventos de forma segura. Considerando duas partes A e B, sempre que uma etapa de comunicação tenha que ser protegida quanto ao não-repúdio, a parte que teria a possibilidade de negar a realização (A) deve registrar a transação na TTP, que gera uma assinatura indicando que a etapa foi registrada. Ao receber esta assinatura da TTP, a outra parte (B) sabe que, se necessário, a TTP poderá ser consultada para obtenção de evidências da realização da ação por parte de A.

2.2.5 Autorização

Em um contexto de segurança de redes, autorização (ou controle de acesso) é a habilidade de limitar e controlar o acesso aos sistemas e aplicações da estação através dos canais de comunicação. Sistemas de controle de acesso podem descrever não somente

quem ou que processo pode ter acesso a um recurso específico, mas também o tipo de acesso que é permitido. A implementação de um mecanismo de autorização pode se basear em diferentes técnicas, dentre as quais se destacam:

- *Access Control List* (ACL): cada recurso ou objeto possui associado uma lista de quais entidades podem realizar quais operações. Uma operação só é permitida se o par entidade / operação está presente na ACL do objeto. Este modelo é análogo ao utilizado na implementação do esquema de permissões de arquivos em sistemas UNIX.

- *Capabilities*: a entidade responsável pelo recurso passa referências dos serviços disponíveis àqueles que podem acessá-los. Estas referências servem como "tickets" que são verificados a cada tentativa de acesso. As referências também podem ser passadas através de delegação. O uso de *capabilities* pressupõe o emprego de um mecanismo para verificação da validade das referências, como assinaturas digitais.

- *Role-Based Access Control* (RBAC - [6]): a entidade responsável associa permissões aos *papéis* que as outras entidades podem assumir no sistema (como por exemplo, "usuário logado" ou "administrador"). Desta forma, o controle de acesso se dá por dois processos independentes: a permissão de uma entidade atuar em um determinado papel e a verificação se entidades de um determinado papel podem acessar um recurso/serviço requisitado. Esta divisão torna RBAC em especial interessante para aplicações P2P de grande escala ([38], [39]), devido à dinamicidade em relação aos integrantes tipicamente existente neste tipo de aplicação.

2.2.6 Auditoria

A auditoria permite análise do funcionamento dos mecanismos de segurança durante ou após a sua execução. A técnica mais simples e difundida com este objetivo consiste na geração de *logs* de execução. Através da análise desse *log*, deve ser possível tanto verificar o correto funcionamento dos mecanismos de segurança, acompanhando a sua execução, quanto possibilitar a identificação de causas e responsáveis por uma falha em algum dos mecanismos. Dependendo da situação, esta análise pode ser feita tanto de forma manual quanto automática, sendo importante haver uma estrutura coerente na geração dos *logs*.

2.2.7 Anonimicidade

A anonimicidade tem por objetivo evitar que uma estação possa ser identificada em uma comunicação com outra estação ou um servidor qualquer. Na forma mais simples, uma estação envia mensagens para o servidor através de um *proxy*, como o Anonymizer.com ([40]). Desta forma, o servidor não consegue identificar diretamente o remetente original das mensagens. O sistema falha caso o *proxy* revele a identidade de um usuário ou se um adversário puder observar o tráfego de entrada e saída do *proxy*. Os servidores podem também facilmente bloquear estes *proxies* centrais. Para evitar este ponto central de falhas, uma estação pode conectar a um servidor através de um conjunto de *relays* ([41]), que transmitem as mensagens entre si até que um deles, definido arbitrariamente para cada conexão, transmite a mensagem ao servidor. O *Anonymous Remailer System* ([42]), o *Onion Routing* [43] e o *Zero-Knowledge's Freedom* [44] utilizam este modelo, provendo anonimicidade através de um conjunto pequeno e fixo de *relays*. Um trabalho mais recente, Tarzan ([45]), utiliza um conjunto variável de *relays*, dentro de uma rede P2P, aumentando as garantias em especial contra uma observação global do tráfego.

A utilização de um mecanismo de anonimicidade pode ser barrada não apenas por aspectos técnicos, mas também pela própria semântica do sistema. Em aplicações que possuem requisitos fortes em termos de segurança, a possibilidade de uma entidade realizar alguma operação de forma anônima pode ser inaceitável. Em especial, percebe-se claro conflito entre um sistema que exige autenticação e a possibilidade de anonimicidade ([29]).

2.2.8 Reputação

O estabelecimento de um mecanismo de reputação entre entidades de um sistema tem por objetivo permitir a um usuário determinar um grau de confiança relativo aos demais nós da rede. Com isso, é possível associar um "risco" ao uso ou prestação de um serviço baseado na reputação da outra parte envolvida. Em redes P2P que não possuam requisitos fortes de autenticação, um esquema de reputação pode atuar como substituto a um sistema baseado em certificados digitais: pode-se confiar em um *peer* simplesmente por seu comportamento no sistema, sem necessidade de mapear o *peer* com a entidade física responsável. Esta característica permite ainda a coexistência de um mecanismo de anonimicidade.

A determinação de um índice de confiança para uma entidade em um sistema P2P passa, em geral, pela consulta (*polling*) dos nós conhecidos na rede ([46], [47], [48]).

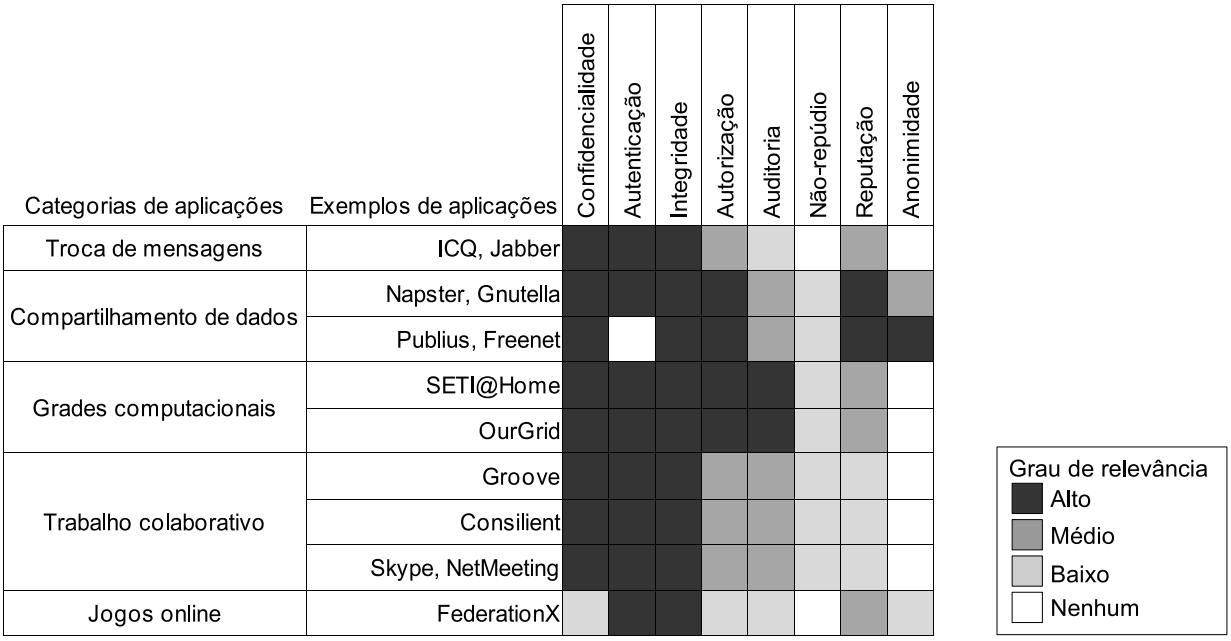


FIGURA 2.4 – Grau de relevância de cada aspecto de segurança para cada tipo de aplicação P2P.

A partir desta pesquisa é feito um cálculo onde o peso da "opinião" dos demais nodos é ponderado pela sua reputação, ou seja, quanto maior a reputação de um nó, mais importante é a sua opinião quanto a outros *peers*.

2.3 Segurança em aplicações Peer-to-Peer

O principal desafio na incorporação de aspectos de segurança a aplicações P2P é a adaptação de mecanismos existentes, tipicamente moldados à realidade cliente-servidor, ao ambiente dinâmico e padrão de uso diferenciado das redes P2P. Diferentes aplicações P2P possuem diferentes necessidades quanto aos aspectos de segurança apresentados na seção anterior. A Figura 2.4 apresenta o grau de relevância que a incorporação de cada aspecto pode ter em cada uma das principais categorias de aplicação P2P. Observa-se que, no geral, confidencialidade, autenticação e integridade são os aspectos mais importantes, seguidos de autorização e auditoria.

Em aplicações de *troca de mensagens*, a autenticação se refere à garantia de que o autor de cada mensagem recebida pelos usuários do sistema partiu de um remetente

legítimo. Da mesma forma, a integridade garante que as mensagens não foram alteradas em trânsito. Ambos os aspectos podem ser considerados bastante importantes em uma aplicação deste tipo. A confidencialidade também pode ser importante, em comunicações sigilosas, estabelecendo-se um canal criptografado entre dois usuários ou cifrando as mensagens individualmente. A autorização, não tão relevante, pode servir para bloquear mensagens de determinados remetentes ou apenas aceitar mensagens de remetentes conhecidos, por exemplo. Por tipicamente não se tratarem de aplicações críticas ou que possam trazer danos aos usuários, a auditoria torna-se dispensável, apesar de facilmente aplicável (mantendo registro das mensagens trocadas). É dispensável também o uso de mecanismos de reputação, uma vez que a autenticação direta dos membros é mais indicada. Por fim, a anonimicidade e o não-repúdio não se aplicam a este tipo de aplicação.

Diferentes aplicações de *distribuição de conteúdo* podem possuir formas bastante diferentes de lidar com os aspectos de segurança. Enquanto em aplicações tradicionais de troca de arquivos (como Gnutella e Napster) os aspectos de segurança podem ser empregados para garantir a "responsabilidade" pelos dados distribuídos, em outras aplicações (onde se destaca a Freenet) a intenção é justamente a contrária: garantir que os dados distribuídos não sejam vinculados à pessoa que criou ou está publicando arquivos. Neste sentido, enquanto autenticação, integridade e, em menor grau, auditoria, podem ser importantes para casos de uso do primeiro grupo de aplicações, a anonimicidade pode (e deve) ser aplicada apenas ao segundo grupo. Já mecanismos de confidencialidade e de reputação, podem ser aplicados a ambos os conjuntos.

Aplicações de *grades computacionais*, principalmente por envolverem o uso de recursos por parte de usuários não vinculados diretamente ao domínio administrativo que os gerencia, podem ser identificadas como tendo os requisitos de segurança mais fortes dentre os grupos de aplicações enumerados. Embora autenticação, integridade, autorização e auditoria se destaquem no que tange a garantir que os recursos não sejam usados por usuários não autorizados ou de forma inadequada, a confidencialidade também pode ser importante no caso da transmissão de tarefas ou resultados sigilosos. O uso de algum mecanismo de reputação pode estar vinculado à autenticação para que, por exemplo, *peers* que executem um maior número de tarefas de outros *peers* tenham uma prioridade maior na execução de suas próprias tarefas remotamente.

Aplicações de *trabalho colaborativo* possuem intrinsicamente uma necessidade de autenticação dos integrantes do grupo de trabalho e de integridade das ações realizadas por cada um deles. A confidencialidade também pode ser um aspecto importante neste tipo

de aplicação, pois não raro o uso de ferramentas de trabalho colaborativo está vinculado a projetos corporativos, onde o sigilo é, em geral, desejado. Dependendo do nível de acesso que os participantes da aplicação podem ter nos recursos dos outros usuários, pode ser importante manter mecanismos de autorização e auditoria. Por sua utilização estar tipicamente vinculada a um grupo de pessoas que se conhecem (fisicamente ou através de uma relação de trabalho remota) as características de não-repúdio e reputação são dispensáveis neste tipo de aplicação. A exemplo do que ocorre em aplicações de troca de mensagens e grades computacionais, o uso de mecanismos para garantir a anonimicidade dificilmente se enquadraria em aplicações de trabalho colaborativo.

Pela sua natureza de utilização, *jogos on-line* possuem requisitos de segurança mais baixos comparado às outras aplicações P2P listadas. Entretanto, mecanismos de autenticação dos jogadores e integridade das mensagens trocadas podem ser importantes para que o jogo tenha um andamento adequado. Mecanismos de confidencialidade, autorização, auditoria e anonimicidade dificilmente são necessários. Um mecanismo de reputação pode ser empregado para, por exemplo, eliminar usuários identificados como *cheaters* (trapaceiros), uma preocupação comum em jogos envolvendo múltiplos jogadores.

Capítulo 3

Trabalhos relacionados

Este capítulo apresenta uma revisão bibliográfica relativa aos principais trabalhos relacionados a esta dissertação. A Seção 3.1 faz considerações sobre bibliotecas de desenvolvimento de aplicações P2P segura, em especial, a Peer-to-Peer Trusted Library. A Seção 3.2 descreve, após uma ambientação sobre o assunto, o modelo de segurança empregado em *web services*, bem como dois exemplos de plataformas baseadas neste paradigma: E-Speak e Globus. Na Seção 3.3 é apresentado o JXTA, plataforma para desenvolvimento de aplicações P2P que incorpora algumas das funcionalidades de segurança comentadas nesta dissertação. É dada ênfase adicional ao funcionamento geral do JXTA devido ao seu uso na fase de implementação desta dissertação, descrita na Seção 5.1. Por fim, na Seção 3.4, são apresentados outros trabalhos relacionados à dissertação que, em geral, visam atender a algum aspecto de segurança específico.

3.1 Bibliotecas de desenvolvimento

A abordagem mais direta para facilitar a incorporação de aspectos de segurança em aplicações P2P é a disponibilização de rotinas que contenham a implementação de algoritmos de segurança aos programadores deste tipo de aplicação. É com este enfoque que foi desenvolvida pela Intel, em 2001, a Peer-to-Peer Trusted Library (PtPTL - [4]).

PtPTL é um *toolkit* de software, código aberto, que provê componentes de segurança especialmente apropriados para aplicações P2P. Estes componentes, disponíveis através de uma interface orientada a objetos, incluem certificados digitais, assinaturas digitais, criptografia simétrica e criptografia assimétrica, oferecendo suporte para incorporação de autenticação, confidencialidade e integridade às aplicações.

Uma das principais preocupações no desenvolvimento de PtPTL foi a portabilidade. PtPTL pode ser utilizado tanto em Windows quanto Linux, e pode ser adaptada para outras plataformas sem necessidade de reestruturação. O principal motivo desta portabilidade é o fato da implementação ser baseada na biblioteca OpenSSL ([49]), que provê o suporte de baixo nível para certificados e criptografia usado no projeto. PtPTL, desta forma, provê uma interface de alto nível para o OpenSSL, tornando mais simples o seu uso considerando as formas de comunicação presentes em redes P2P. O projeto OpenSSL consiste em um esforço colaborativo para desenvolver um *toolkit* robusto, completo e de código aberto que implemente os protocolos SSL e TLS, e que represente uma biblioteca de criptografia para uso geral.

Outro projeto que oferece uma API de alto nível para utilização de algoritmos relativos à segurança de aplicações P2P é o JXTA ([8]). Entretanto, o projeto tem uma abrangência maior do que a simples disponibilização de um *toolkit* de segurança, sendo tratado em maiores detalhes na Seção 3.3.

3.2 Web services

A tecnologia de web services, até por ser bastante empregada em operações comerciais, estabeleceu um modelo maduro de segurança. Considerando aspectos como a heterogeneidade e fraco acoplamento entre as entidades que formam o sistema, a tecnologia de *web services* pode ser comparada com redes P2P. Esta seção apresenta uma descrição geral da tecnologia de *web services*, descrevendo características do modelo de segurança empregado.

Web services provêem um meio para comunicação entre aplicações sobre a Internet ([50]). Uma vez que a Internet compreende aplicações e plataformas heterogêneas, *web services* naturalmente trazem características de interoperabilidade e extensibilidade a essas várias aplicações e plataformas ([38]). O W3C ([51]) trabalha no estudo de requisitos e na definição de uma arquitetura padrão para esta tecnologia, mantendo a especificação em [52]. Para permitir esta interoperabilidade de arquiteturas e modelos de processamento, *web services* evoluíram como uma combinação de diversos padrões tecnológicos, como XML (eXtensible Markup Language), WSDL (Web Service Description Language - [53]), UDDI (Universal Description, Discovery and Integration - [54]) e SOAP (Simple Object Access Protocol - [55]).

A arquitetura de *web services* é baseada na interação entre três componentes princi-

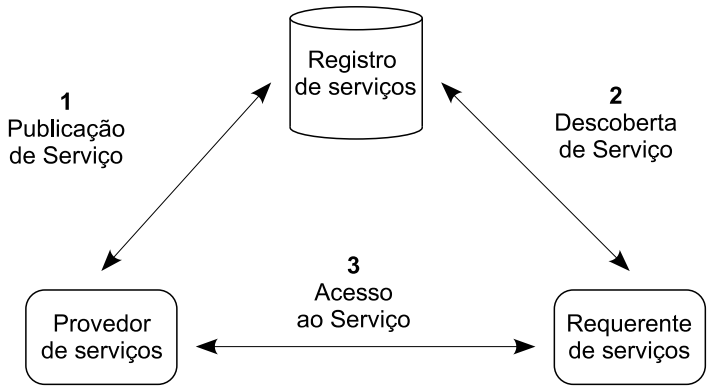


FIGURA 3.1 – Modelo conceitual de Web services.

pais (Figura 3.1): provedor de serviços (*service provider*), registro de serviços (*service registry*) e requerente de serviços (*service requestor*). Estes componentes interagem usando operações de publicação (*publish*), busca (*find*) e ligação (*bind*). O provedor de serviços é a entidade que provê acesso aos *web services* e publica a descrição dos serviços (1) em um registro de serviços. A publicação de um serviço é realizada através de sua descrição seguindo o padrão WSDL. O requerente de serviços localiza, através da descrição, o serviço desejado no registro de serviços (2) e usa esta informação para realizar a ligação com os provedores escolhidos. Nesta etapa, o requerente recebe, além da informação de como acessar o provedor, informações de como a requisição ao serviço deve ser realizada. Por exemplo, o nome do método a ser acessado e os parâmetros que devem ser informados (bem como os seus tipos). De posse destas informações, o serviço pode ser diretamente acessado (3).

Um dos principais aspectos da padronização de segurança em *web services* (Web Services Security - WSS - [56]) é a adição de características de segurança (em especial autenticação, integridade e confidencialidade) ao SOAP, protocolo empregado na comunicação entre entidades. Não é determinado o uso de algoritmos específicos (por exemplo, de cifragem), mas um suporte geral que permite o uso de diferentes técnicas. A especificação de quais requisitos de segurança devem ser obedecidos é realizada individualmente para cada serviço por parte do provedor. Ao receber uma requisição, o provedor de serviços primeiramente verifica o atendimento dos requisitos especificados para acesso ao serviço, para então proceder com o atendimento da requisição.

Atualmente, são várias as plataformas de *web services* disponíveis, ao mesmo tempo

em que é dada grande importância à padronização dos seus elementos, para permitir a interoperabilidade entre as mesmas. A seguir, são apresentadas duas plataformas baseadas em *web services* com especial relação a este trabalho: E-Speak e Globus.

3.2.1 E-Speak

O E-Speak ([57]), projeto iniciado em 1995 pela HP, aparece como uma plataforma cujas técnicas contribuíram para dar definir o estado atual da tecnologia de *web services*. O principal objetivo da criação do E-Speak foi a sua utilização em aplicações de comércio eletrônico. O E-Speak já traz a idéia de separar os serviços disponibilizados dos meta-dados que regem a sua utilização, uma das principais características do modelo de *web services* padronizado atualmente (com o uso de WSDL). Como principal diferença ao modelo atual de *web services*, E-Speak possui o conceito de vocabulários, documentos preparados independentemente pelos participantes do sistema para definir os tipos de dados e serviços publicados. Uma vez que estes vocabulários preparados independentemente podem apresentar definições conflitantes, faz-se necessária a utilização de um mecanismo específico para lidar com este tipo de conflito. Atualmente em *web services* este tipo de definição de tipos é realizada através de padrões estabelecidos globalmente pelo W3C, garantindo a não-ambiguidade na troca de mensagens e busca por serviços.

Em [3], os autores apresentam o E-Speak como uma plataforma segura para aplicações P2P. Argumenta-se que, enquanto aplicações P2P em geral são direcionadas para a prestação de apenas um ou dois tipos de serviços (por exemplo, compartilhamento de arquivos ou troca de mensagens), enquanto E-Speak poderia ser usado para sistemas P2P onde há uma gama mais abrangente de serviços disponibilizados. Em termos de aspectos de segurança, destaca-se que E-Speak provê:

- Autenticação: a autenticação entre os membros do sistema é realizada utilizando-se um mecanismo similar ao PGP, que dispensa a necessidade de uma entidade certificadora central;
- Confidencialidade: a comunicação segura entre os *peers* é provida através de SLS (*Session Layer Security*), que estende SSL (*Secure Socket Layer*). As partes utilizam SLS no *handshake* inicial, onde é utilizado o algoritmo Diffie-Hellman para estabelecimento de uma chave comum para cifragem das informações. Nesta etapa é também definido qual o algoritmo de cifragem que será adotado na comunicação;

- Autorização: para controlar o acesso aos recursos, E-Speak utiliza *capabilities*, devidamente assinadas para garantir a sua validade. Conforme explicado na Seção 2.2.5, *capabilities* são distribuídas pelo responsável pelo recurso às entidades que são autorizadas a acessá-lo.

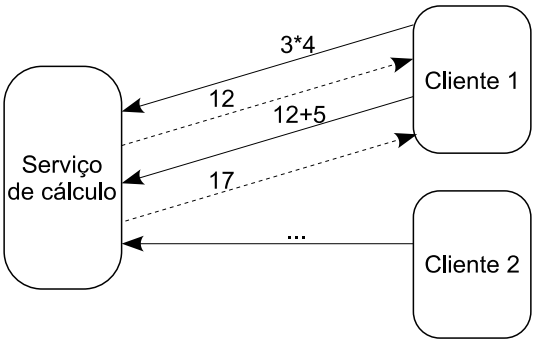
Atualmente, o projeto E-Speak foi absorvido pela segunda geração de ferramentas para *web services* da HP.

3.2.2 Globus

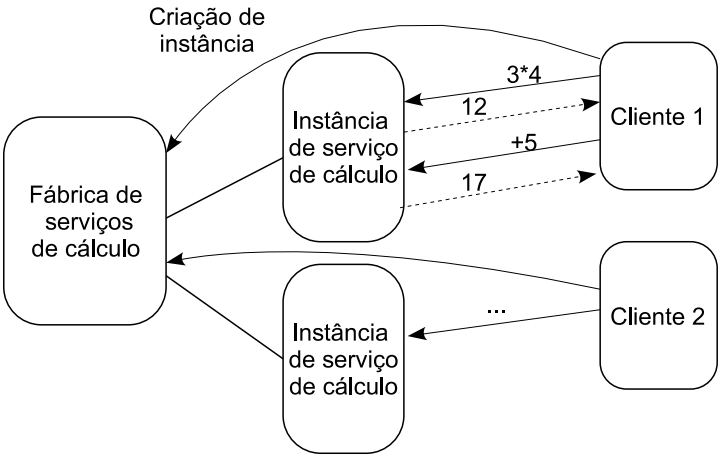
Apesar de se adequar em uma vasta gama de aplicações, *web services* possuem algumas limitações que dificultam o seu uso em aplicações mais complexas, como de grades computacionais, onde o termo grade se refere a "sistemas e aplicações que integram e gerenciam recursos e serviços distribuídos através de múltiplos domínios administrativos" ([58]). Buscando atender a esta limitação, foi estabelecida a idéia de *grid services*, que correspondem a *web services* complementados e especializados em função de aplicações de grade computacional.

A principal característica introduzida em *grid services* (e não presente em *web services*) é a padronização de serviços com estado (*stateful*), conforme ilustrado na Figura 3.2. Enquanto em *web services* a execução de diferentes tarefas inter-relacionadas implica em retransmissões de resultados para o provedor do serviço, utilizando *grid services* é possível criar uma instância de um serviço que armazena as informações relevantes de estado. Por exemplo, a execução de uma cadeia de cálculos (composta de diversas requisições) implicaria, no caso de *web services*, a transmissão, juntamente com cada requisição, dos resultados obtidos até o momento. No caso da Figura 3.2 (a), o Cliente 1 deve retransmitir o resultado da primeira operação (12) para que outra operação possa ser realizada sobre este valor. Já no caso de um *grid service*, a própria instância do serviço armazena o estado do processamento, mesmo que este seja relativo a uma série de requisições. No caso da Figura 3.2 (b), o resultado da primeira operação não precisa ser retransmitido quando da realização de uma segunda operação sobre este valor. *Grid services* podem ser criados e destruídos conforme a demanda (acessando a "fábrica" de serviços correspondente), sendo toda a gerência do ciclo de vida de um *grid service* regida de forma padronizada.

No contexto de *Grid services*, se destacam duas padronizações (OGSA e OGSI) e o Globus Toolkit. O Open Grid Services Architecture (OGSA), desenvolvido pelo Global Grid Forum, busca definir uma arquitetura comum, padronizada e aberta para aplicações baseadas em grades. O OGSA padroniza os serviços existentes em uma aplicação de



(a) Web services (sem estado)



(b) Grid services (com estado)

FIGURA 3.2 – Diferença de funcionamento entre *web services* e *grid services* quanto a presença ou não de informações de estado.

grade (por exemplo, serviços de segurança e serviços de gerência de tarefas e de recursos) especificando uma série de interfaces para estes serviços. OGSA por si só não não descreve completamente *grid services*, apenas define quais as características de um Grid service (que o diferencia de um *web service*). Para atingir um maior nível de detalhamento nessa definição, surgiu o Open Grid Services Infrastructure (OGSI, também desenvolvido pelo Global Grid Forum), que traz uma especificação formal e técnica do que é um Grid service. Em suma, *grid services* são definidos pelo OGSA e especificados pelo OGSI. O Globus ([59]) é um *toolkit* de software, desenvolvido pela Globus Alliance, que pode ser usado para programar aplicações baseadas em grade. A terceira versão do *toolkit* (GT3) inclui uma implementação completa do OGSI, além de uma série de outros serviços, programas e funcionalidades em geral.

Globus oferece mecanismos de autenticação, confidencialidade, integridade, controle de acesso e auditoria. A implementação destas características de segurança se baseiam em uma estrutura específica, o GSI (Grid Security Infrastructure). A exemplo do que ocorre em *web services* tradicionais, GSI prevê que os requisitos de segurança dos serviços oferecidos são disponibilizados utilizando interfaces WSDL. Isto permite que os clientes possam descobrir dinamicamente quais credenciais e mecanismos são necessários para a utilização de determinado serviço ([58]). Adicionalmente, é empregado o WS-Security ([56]) para a troca segura de mensagens entre os integrantes do sistema. O GSI prevê a disponibilização de diferentes serviços de segurança, que possibilitam a comunicação entre entidades heterogêneas no que tange aos mecanismos de segurança empregados. Por exemplo, um serviço de conversão de credenciais permite que participantes que possuem um tipo de certificado digital (por exemplo, Kerberos - [60]) possa ser autenticado através de outro (por exemplo, X.509 - [61]).

Os mecanismos de autenticação e autorização em Globus possuem em especial funcionalidades bastantes avançadas. É possível realizar uma autenticação inter-domínios através do mapeamento de um usuário global para um contexto local. Por exemplo, um usuário pode ter associado, dentro de cada um dos domínios administrativos que compõem a grade, uma conta (*login*) diferente. Outra característica importante é a possibilidade de delegação de credenciais. Esta operação é realizada através de certificados digitais específicos, assinados pela entidade delegadora dando direito a outra realizar alguma operação em seu nome. Esta característica se torna em especial interessante considerando o modelo de serviços com estado empregado no Globus. Um cliente pode autorizar uma instância de um serviço criado a acessar os recursos necessários para a realização das

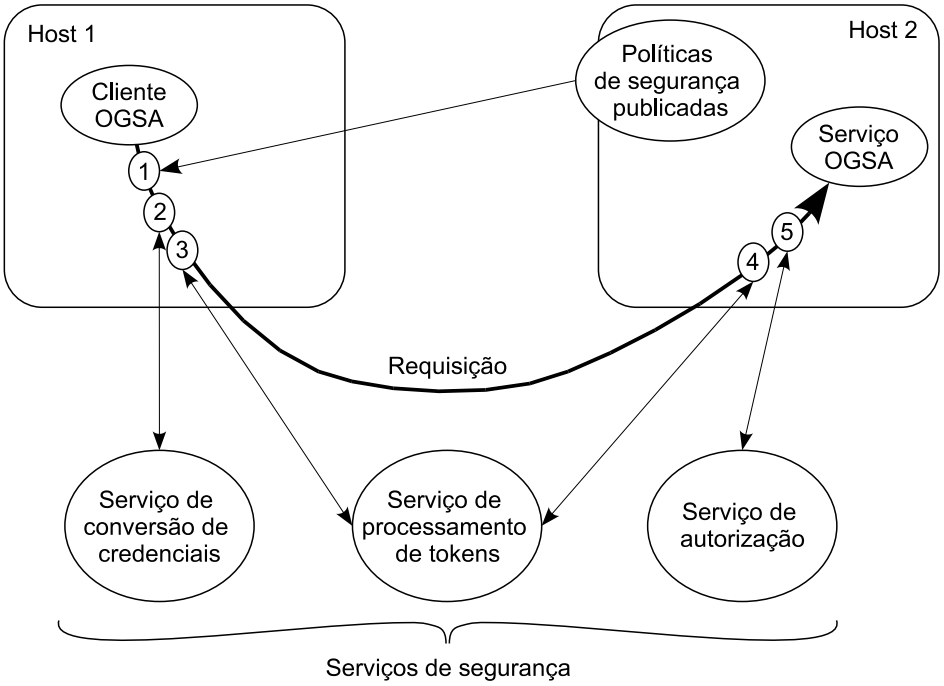


FIGURA 3.3 – Exemplo de acesso a um serviço utilizando GSI.

tarefas (acessando outros serviços, por exemplo) em seu nome.

A Figura 3.3 apresenta um exemplo simples de funcionamento do GSI ([58]). Primeiramente, o cliente verifica (1) as políticas de segurança do serviço alvo para determinar quais mecanismos e credenciais são necessários para submeter uma requisição. Em seguida, o cliente determina (2) quais credenciais não estão preparadas, e contacta um serviço de conversão de credenciais para converter as credenciais existentes para o formato, mecanismo ou autoridade de certificação exigidos. O cliente então acessa (3) um serviço de processamento e validação de *tokens* de autenticação (representação das credenciais no formato XML) para adaptar a mensagem de requisição ao formato exigido no provedor. Este serviço dispensa as aplicações da necessidade de lidar com detalhes de mecanismos específicos. No lado do servidor, também é acessado (4) o serviço de processamento e validação de *tokens* para analisar as informações enviadas pelo cliente. No exemplo, ambas as partes utilizam o mesmo serviço, mas poderiam utilizar instâncias diferentes. Após a autenticação e a determinação da identidade e atributos do cliente, estas informações, juntamente com a requisição, são repassadas (5) para um serviço de autorização, para aprovação ou não baseado nas políticas estabelecidas. Caso todas as

etapas sejam completadas com sucesso, a aplicação responsável pelo serviço alvo pode, sabendo que as medidas de segurança foram tomadas, realizar as operações relativas à requisição recebida.

3.3 JXTA

O JXTA ([8], [62]) é um projeto que visa estabelecer um conjunto de protocolos, independentes de implementação, que possibilitem a criação de uma estrutura P2P de uso geral, que possa ser empregada por diferentes aplicações P2P. Os principais objetivos do projeto são ([8]):

- Interoperabilidade: Em geral, sistemas P2P são construídos para atender a um tipo específico de serviço. Por exemplo, Gnutella provê compartilhamento de arquivos, e ICQ provê troca de mensagens (*instant messaging*). Dado as características diversas dos serviços e a falta de uma estrutura P2P subjacente comum, desenvolvedores de sistemas P2P tendem a criar sistemas incompatíveis. Grande parte do desenvolvimento da aplicação acaba representando esforço duplicado para a criação de funcionalidades comumente utilizadas em todos os sistemas P2P. Além disso, para um *peer* participar de diferentes comunidades organizadas por diferentes implementações, ele deve possuir suporte específico a cada uma destas implementações. JXTA busca criar esta camada subjacente comum para diferentes aplicações P2P, permitindo integrá-las e facilitando o seu desenvolvimento;

- Independência de plataforma: Muitos sistemas P2P atuais disponibilizam suas funcionalidades através de um conjunto de APIs que são oferecidas por um sistema operacional em particular usando um protocolo de rede específico. No caso de um desenvolvedor desejar oferecer um mesmo serviço em diferentes sistemas ou ambientes de programação, faz-se necessária a criação de diferentes implementações. A padronização existente no JXTA independe de linguagem de programação, sistema operacional ou protocolo de comunicação específicos;

- Ubiqüidade: A arquitetura de JXTA foi desenvolvida de forma que seja possível implementá-la em qualquer dispositivo computacional, incluindo sensores, PDAs, computadores pessoais e sistemas de armazenamento. A hierarquia dos elementos que compõem a rede existente no JXTA permite reduzir a complexidade das tarefas

realizadas nos nodos folhas, concentrando funcionalidades em nodos centrais com maior conectividade e poder de processamento.

A busca de informações, um dos principais aspectos de uma rede P2P, é realizada no JXTA utilizando um mecanismo descrito em [62] que, embora descentralizado e independente de uma entidade central de indexação, utiliza um modelo onde *peers* centrais são responsáveis por rotear, entre si, as mensagens de busca. Aos nodos folhas, cabe apenas publicar os serviços ou recursos disponíveis em algum nodo central. Este arquitetura corresponde ao modelo descentralizado não estruturado heterogêneo, descrito na Seção 2.1.

Apesar de ser um projeto liderado pela Sun, desde a sua criação houve uma grande preocupação em torná-lo aberto ([63]), com colaboração da comunidade científica e de software livre. Esta característica incentivou a criação de um grande número de ferramentas e projetos ligados ao JXTA ([64]). Apesar de a implementação de referência (baseada em Java) ser a mais utilizada e difundida, já existem iniciativas de desenvolvimento da plataforma JXTA em outras linguagens, como C++ e Python.

Funcionamento básico do JXTA

No mais alto nível de abstração, JXTA é um conjunto de protocolos ([8]), cujo funcionamento é definido de forma independente de linguagem ou protocolo de transporte subjacente. Alguns conceitos são em especial importantes para o entendimento da dinâmica de funcionamento do JXTA: *peer*, grupo de *peers*, mensagens, *pipes* e anúncio. A Figura 3.4 apresenta um exemplo de cenário de uso do JXTA que contempla estes conceitos.

Um ***peer*** é qualquer entidade que consegue se comunicar através dos protocolos JXTA correspondentes. É importante ressaltar que um *peer* não necessariamente precisa implementar todos os seis protocolos JXTA definidos a seguir, mas apenas aqueles da qual depende sua funcionalidade no sistema P2P.

O JXTA permite o estabelecimento de **grupos de *peers***, tipicamente compostos de *peers* cooperativos executando uma mesma aplicação. Não existe limite para o número de grupos da qual um *peer* faz parte, e o grupo ***World Peer Group*** é composto por todos os *peers* que fazem parte da rede.

Em JXTA, as **mensagens** foram estabelecidas de forma a poderem ser usadas sobre uma camada de transporte assíncrona, não-confiável e unidirecional. Elas são implementadas através de datagramas que contém um envelope, contendo origem e destino, e uma pilha de cabeçalhos de protocolos além dos dados sendo transmitidos.

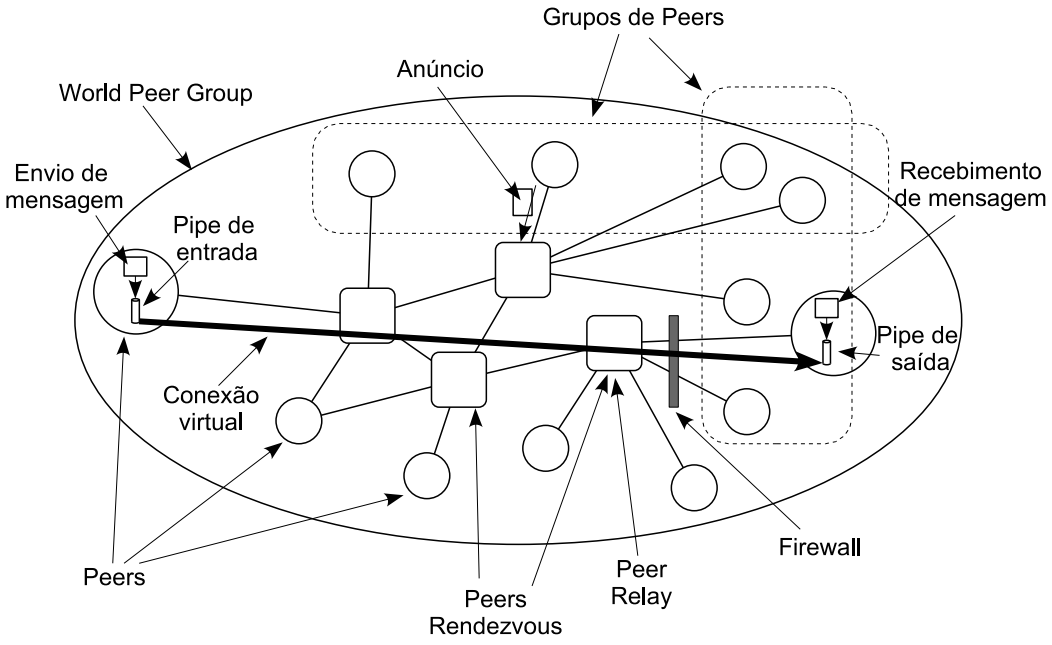


FIGURA 3.4 – Elementos de uma rede JXTA.

Pipes são canais de comunicação assíncronos que devem ser estabelecidos entre as partes para enviar e receber mensagens. Cada *pipe* é unidirecional, havendo, portanto, ***pipes* de entrada** e ***pipes* de saída**. Cada "ponta" (*endpoint*) do *pipe* pode estar ligada a um ou mais *peers*, permitindo comunicação *unicast*, *broadcast* ou *multicast*. JXTA oferece ainda o conceito de *pipe* seguro, que estabelece uma comunicação com autenticação, integridade e confidencialidade entre as parte comunicantes.

Um **anúncio** (*advertisement*) é um documento XML que nomeia, descreve e publica a existência de uma entidade, como um *peer*, um grupo de *peers*, um *pipe* ou um recurso qualquer (por exemplo, arquivo compartilhado ou serviço disponibilizado). Para poderem ser localizados na rede, os recursos devem ser publicados explicitamente através de um anúncio apropriado.

As funcionalidades de JXTA estão presentes na especificação de seus protocolos, comentados a seguir.

- Protocolo de descoberta (*Peer Discovery Protocol*): mecanismo pela qual um *peer* pode divulgar seus recursos, e descobrir outros *peers*, grupos de *peers* ou recursos de outros *peers*.

- Protocolo de resolução (*Peer Resolver Protocol*): mecanismo pelo qual um *peer* pode enviar uma consulta para um, vários ou todo um grupo de *peers*, e receber uma resposta (ou múltiplas respostas). A verificação de qual consulta originou as respostas recebidas é realizada através de um identificador único, incluso no corpo da mensagem.

- Protocolo de informações de *peer* (*Peer Information Protocol*): mecanismo pela qual um *peer* pode obter informações de estado e capacidades de outros *peers*. Por exemplo, é possível enviar uma mensagem de *ping* para verificar se o *peer* está ativo, ou verificar o seu tempo de funcionamento sem interrupção.

- Protocolo de ligação de pipe (*Pipe Binding Protocol*): permite a um *peer* estabelecer um pipe (canal virtual de comunicação) com um ou mais *peers*. A ligação entre as entidades é realizada quando o *pipe* é durante a operação de abertura (*open*). Após a ligação, podem ser realizadas operações de *send* (de um dos lados) e *receive* (do outro lado).

- Protocolo de roteamento (*Endpoint Routing Protocol*): implementado nos *peer routers*, permite aos *peers* da rede requisitar uma rota para o envio de mensagens a outro *peer*. Por exemplo, quando dois *peers* não estão diretamente conectados, estão utilizando protocolos de transporte diferentes ou quando estão separados por *firewalls* ou serviços de NAT, o *peer router* responde a requisição com a rota que pode ser utilizada. Em especial, conforme exemplificado na Figura 3.4, esta rota pode conter *peers relay*, que possibilitam acionar conexções com *peers* barrados por *firewalls* ou NATs. Neste caso, as mensagens são enviadas para o *peer relay*, que as repassa ao *peer* destino, que não poderia ser alcançado diretamente. Qualquer *peer* pode se tornar um *peer router* através da instanciação do protocolo de roteamento.

- Protocolo de *rendezvous* (*Rendezvous Protocol*): usado por um *peer* para conectar a um *peer rendezvous*, que concentra algumas funções chave dentro da rede. Em especial, estes *peers* servem como referência para o ingresso na rede JXTA: listas de IPs de nodos *rendezvous* são acessíveis através de *sites* na Internet, possibilitando a conexão de novos *peers*. Além desta tarefa, *peers rendezvous* realizam a indexação do conteúdo de nodos vizinhos, otimizando o processo de busca.

Segurança em JXTA

O estabelecimento de mecanismos de segurança está presente desde o processo inicial de planejamento do JXTA. A estratégia adotada foi a generalidade no nível conceitual, permitindo a extensão através do uso de mecanismos considerados mais adequados para cada aplicação (ou mesmo algoritmos ainda por vir). Por exemplo, as mensagens suportam a adição de campos para credencias de autenticação ou verificadores sem qualquer alteração nos protocolos.

A implementação dos *pipes* seguros é realizada sobre TLS (*Transport Layer Security* - [65]). TLS cria um canal de comunicação entre as partes, que oferece confidencialidade, integridade e autenticação. A confidencialidade é alcançada através do uso de algoritmos de criptografia simétrica - em especial, *Triple* DES e RC4 - para a cifragem dos dados. As chaves empregadas são únicas para cada conexão, e são definidas na fase de estabelecimento da conexão. A integridade e a autenticação são garantidas através do uso de funções *hash* não reversíveis, como MD5 e SHA1. A fase de estabelecimento de conexão (*handshake*) emprega criptografia assimétrica (RSA) para autenticação mútua das partes e para a definição, de forma segura, da chave simétrica a ser empregada no decorrer da conexão.

O modelo de certificados digitais empregado por padrão no JXTA é o X.509 ([61]). Certificados digitais permitem associar uma chave pública a uma entidade de forma garantida. Ou seja, consultando um certificado digital emitido por uma entidade confiável, um *peer* pode ter certeza que um dado decodificado com a chave pública correspondente foi de fato cifrado usando a chave privada legítima da entidade responsável pela transmissão. A entidade confiável que gera os certificados é a autoridade de certificação (Certification Authority - CA), conforme explicado na Seção 2.2.2.

Adicionalmente, a implementação Java de JXTA possui uma biblioteca, *JXTA-Crypto,* com mecanismos de segurança que oferecem as funcionalidades de criptografia (RSA, RC4), de cálculo de funções *hash* (MD5, SHA1) e código de autenticação de mensagens (MAC) de forma isolada, não vinculada a uma conexão TLS. Seu uso é especialmente útil quando do uso de aspectos de segurança baseados em mensagens individuais, sem empregar *pipes* seguros.

Além dos mecanismos de segurança relativos à comunicação entre os *peers*, JXTA protege a utilização dos *peers* com um ID e uma senha, que são usados para decodificar a chave privada para o ambiente pessoal do usuário. Esta característica age como uma primeira defesa contra um atacante local, com acesso físico à máquina que executa o *peer*

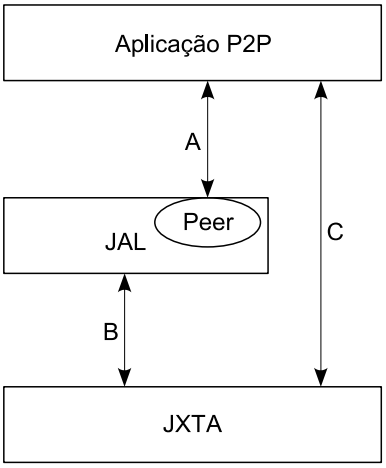


FIGURA 3.5 – Interação entre a aplicação P2P, JAL e JXTA.

JXTA.

JXTA Abstraction Layer

A JXTA Abstraction Layer (JAL - [66]) é uma biblioteca que tem por objetivo principal facilitar o desenvolvimento de aplicações P2P sobre JXTA. O JAL abstrai vários aspectos da arquitetura do JXTA, oferecendo ao programador uma interface mais simples e mais próxima daquela existente quando se faz uma aplicação cliente-servidor.

A Figura 3.5 ilustra a forma de interação entre a aplicação P2P, JAL e JXTA. O principal ponto de interação da aplicação com o JAL (A) é a interface `Peer`, que oferece métodos para localização de *peers*, criação e participação de grupos, envio (para outro *peer* ou para todo um grupo) e recebimento de mensagens. A interface `Peer` possui uma implementação padrão oferecida pelo JAL, a classe `EZMinimalPeer`. A comunicação de JAL com JXTA (B) se dá através da criação e utilização de *pipes*. Sempre que o envio de uma mensagem for ordenado pela aplicação, é criado (ou apenas acessado, caso já exista) um *pipe* com o destino da mensagem. Ou seja, a gerência dos *pipes* necessários é realizado pelo JAL, sem necessidade de intervenção da aplicação. Apesar de JAL possuir uma interface completa no sentido de oferecer as funcionalidades tipicamente empregadas em aplicações P2P, aspectos, em especial de configuração do *peer*, podem ser acessados pela aplicação diretamente pela API do JXTA (C). Um aspecto a ser ressaltado é o fato de JAL, apesar de apresentar uma interface diferenciada, emprega os protocolos JXTA,

herdando todas as suas facilidades (como transparência de rede e habilidade de transpor *firewalls*).

3.4 Outras iniciativas

Além das iniciativas que buscam disponibilizar ferramentas para criação de aplicações P2P seguras, outros trabalhos se destacam em relação a esta dissertação.

Em [67] são identificadas técnicas para controlar a entrada de membros nos grupos de *peers*, fazendo com que apenas *peers* autorizados participem do sistema. São definidos dois elementos básicos para este tipo de mecanismo: o *Group Charter* e o *Group Authority*. O *Group Charter* é um documento digital que contém claramente as regras para aceitação no grupo. Na forma mais simples, este documento pode ser uma *Access Control List* (ACL), onde os *peers* que podem fazer parte do grupo são listados. Apesar de um esquema baseado em ACL ser aplicável para diversos tipos de sistema, uma política de admissão mais dinâmica pode ser necessária em outros. O *Group Authority* (GAUTH) é a entidade responsável por garantir a execução do *Group Charter*, e capaz de emitir certificados que indiquem que membro está autorizado a participar do grupo. O GAUTH pode ser representado, por exemplo, por uma autoridade de certificação (*Certification Authority* - CA) ou pelo próprio grupo de *peers*, que podem realizar uma votação para permitir ou não a entrada de cada novo membro. O estudo faz ainda consideração sobre possíveis procedimentos para admissão de *peers*, bem como técnicas de assinatura empregáveis neste contexto.

O P2P-Role ([39]) é uma arquitetura de controle de acesso para redes P2P. O modelo adotado é o RBAC (*Role Based Access Control*), onde as permissões não são designadas diretamente aos usuários, mas indiretamente através de papéis. Apesar de ser possível empregar as idéias em uma rede puramente P2P, o artigo enfoca o uso em uma rede híbrida, com uma entidade central responsável pela indexação dos recursos. O controle ao acesso aos recursos é realizado localmente em cada *peer*, que possui um banco de dados que representa o modelo RBAC, onde se destacam as tabelas de *usuário*, *papel* e *direito*. A Figura 3.6 apresenta o funcionamento do processo de busca no P2P-Role. Cada participante indexa os recursos locais no computador central (1). O acesso a um recurso é precedido por uma consulta ao computador central (2), que repassa o endereço dos participantes que possuem o recurso desejado (3). De posse destes endereços, o requerente pode contactar diretamente o participante que provê o recurso (4), através de uma co-

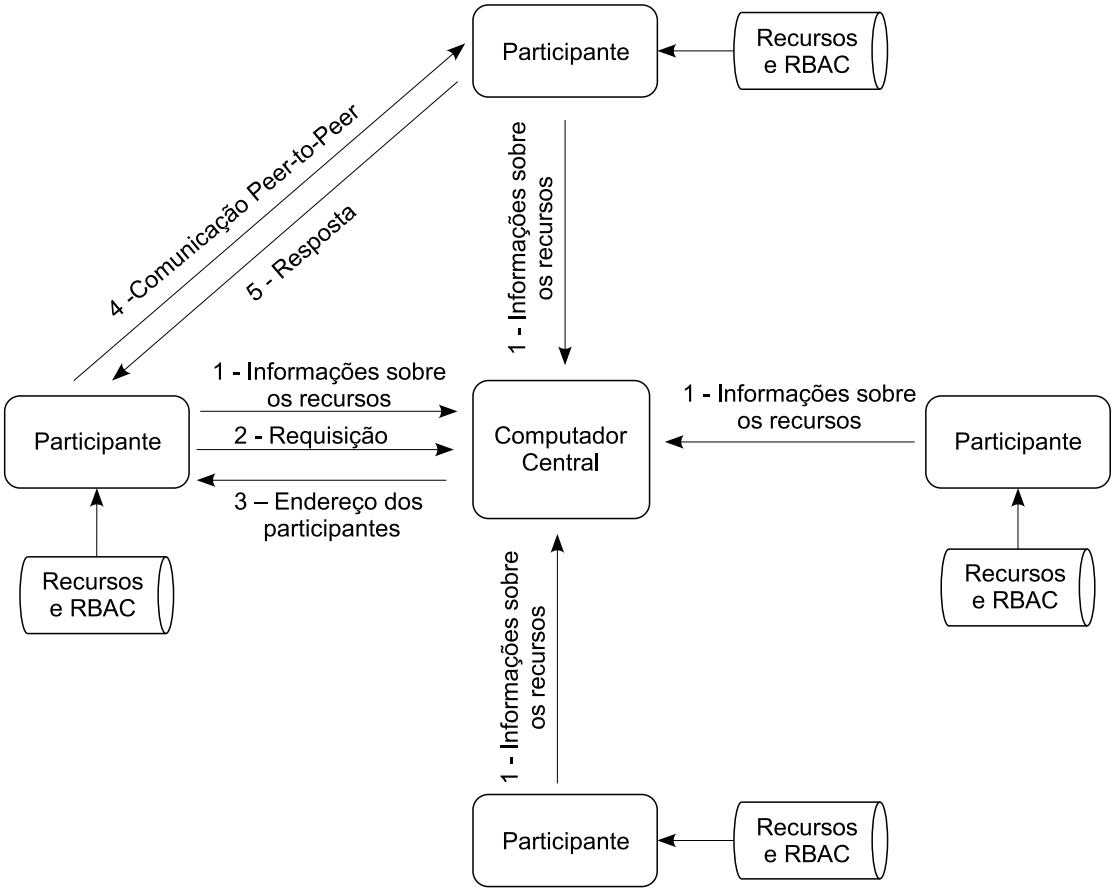


FIGURA 3.6 – Dinâmica de acesso a recursos no P2P-Role.

municação P2P. Nesta etapa, o provedor consulta o módulo RBAC (isolado da aplicação) que irá ou não garantir o acesso do requerente ao recurso desejado. A implementação do protótipo para a arquitetura P2P-Role foi realizada utilizando diretamente a API de redes da linguagem Java. Entretanto, um dos objetivos do estabelecimento da arquitetura P2P-Role é a sua adoção em outras plataformas e projetos que contemplem o tema P2P (como JXTA, por exemplo).

Em [38] é apresentada uma arquitetura de controle de acesso baseada em RBAC para redes P2P, ilustrada na Figura 3.7. É proposto e implementado um *middleware* genérico que pode estar centralizado em uma máquina ou distribuído em vários nodos. Sua função é atuar como um diretório de serviços que indexa os recursos da rede P2P, similar ao registro de serviços em *web services*. Suas funcionalidades estão divididas entre 5 componentes principais dispostos em camadas. O *Communication Gateway* atua como

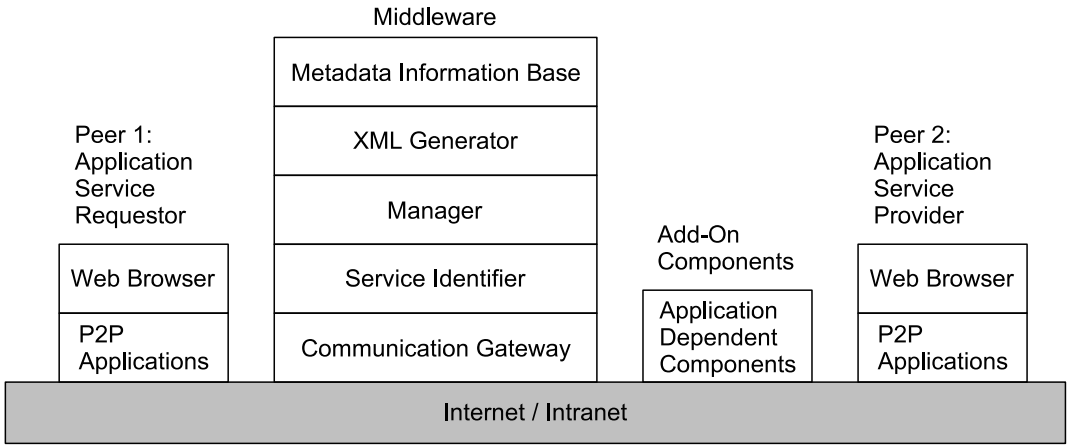


FIGURA 3.7 – Arquitetura apresentada em [38].

interface de acesso ao *Service Indentifier*, que processa as requisições para os serviços, mantendo documentos WSDL (Web Service Description Language). Estes documentos descrevem os tipos de serviços gerenciados pelo *middleware*. O *Manager* é a entidade responsável por gerenciar e controlar os recursos dos *peers*. Por fim, a *Metadata Information Base* armazena as informações dos serviços disponíveis, através de documentos XML padronizados, gerados pelo *XML Generator*. Adicionalmente, o artigo destaca a possibilidade de adição de componentes dependentes de aplicação, isolados do *middleware* genérico (*Add-On Components*). O acesso ao sistema P2P é realizado, do ponto de vista do usuário, através de um navegador Web, dotado de *plugins* específicos para a aplicação P2P subjacente. Um *peer* (Peer 2) que deseja disponibilizar algum serviço (ou recurso), acessa o *middleware* e efetua o registro. As políticas de acesso ficam centralizadas no próprio *middleware*. Quando um outro *peer* (Peer 1) desejar acessar o serviço, ele consulta o *middleware*, que responde a consulta com a informação do *peer* que provê o serviço requisitado (Peer 1). As as políticas referentes ao Peer 2 são repassadas diretamente ao Peer 1, de forma que este último posse verificar se a requisição que irá partir do Peer 2 deve ou não ser atendida.

3.5 Considerações gerais

A revisão bibliográfica revelou a existência de importantes iniciativas relacionadas com este trabalho. A disponibilização de bibliotecas de programação que incorporam

mecanismos de segurança tipicamente usados, apesar de útil, mantém a necessidade da prototipação da lógica de utilização desses mecanismos dentro da aplicação. Questões como quando a comunicação deve ser realizada de forma segura ou quais *peer* podem acessar de que forma os recursos disponibilizados acabam por dificultar o desenvolvimento, prejudicando o isolamento dos aspectos de segurança e a flexibilidade da aplicação criada. Além disso, o fato de não haver uma plataforma subjacente que lide com aspectos de transporte implica na necessidade de prototipação adicional para lidar com aspectos como a existência de *peers* cuja comunicação direta é barrada por *firewalls* ou serviços de NAT.

As soluções baseadas em *web services*, apesar de formarem uma arquitetura bastante genérica e abrangente para a segurança de sistemas fracamente acoplados, tendem a trazer uma grande complexidade na implantação da solução. Um minucioso processo de especificação das políticas empregadas em cada entidade que provê o serviço é necessário, tornando o seu uso pouco indicado para aplicações dinâmicas. No caso do Globus, várias funcionalidades importantes ainda não foram completamente integradas ao *middleware*, cabendo a aplicação lidar com os mesmos.

JXTA, apesar de ser uma plataforma bastante completa para implementação de aplicações P2P e incluir diversas funcionalidades relativas à segurança, possui um problema similar ao das bibliotecas de desenvolvimento: o não isolamento da utilização e configuração dos mecanismos de segurança em relação à aplicação. Apesar de boa parte da base para a implementação de uma aplicação P2P segura estar disponível, ainda cabe ao programador da aplicação lidar com cada um dos aspectos a serem atendidos ([68]). Além disso, o JAL (JXTA Abstraction Layer) não possui integração com os mecanismos de segurança oferecidos pelo JXTA, prejudicando a implantação de mecanismos de segurança em aplicações baseadas nesta API.

Outros estudos, como [38] e [39], têm em mente a modularidade, mas apenas alguns aspectos fazem parte da solução, como, no caso, autenticação e autorização. Considerando que aplicações P2P podem possuir requisitos no que tange a diversos aspectos de segurança (conforme apresentado na Seção 2.3), diferentes soluções independentes teriam que ser empregadas, o que pode representar um grande empecilho.

A presente dissertação descreve uma solução que visa atender às dificuldades supracitadas. É apresentada uma arquitetura genérica que contempla aspectos de reconfigurabilidade, modularidade e isolamento da aplicação. Além do modelo teórico, foi realizada uma implementação para uso sobre JXTA e JAL, que pode servir como camada de segurança para utilização em aplicações existentes ou no desenvolvimento de novas aplicações

P2P. O fato da implementação se basear sobre JXTA permite usufruir das características já implementadas, como adequação à estrutura de segurança normalmente utilizada na Internet (baseada em *firewalls* e NAT).

Capítulo 4

Arquitetura Proposta

4.1 Motivação

A implantação de aspectos de segurança em aplicações P2P ainda é um problema bastante presente. Apesar de já existirem soluções com este objetivo (Capítulo 3), nenhuma delas permite que a incorporação seja feita de forma simples, sem acarretar complexidade adicional no código da aplicação. Este trabalho apresenta uma arquitetura que atende a esta demanda, tendo como principais metas as seguintes características:

- Encapsulamento: isolar a implementação dos aspectos de segurança do restante do sistema, de forma a não implicar complexidade adicional ao código da aplicação e permitir a adição simples dos aspectos de segurança a aplicações já existentes;
- Modularidade: em termos de segurança computacional, existe um compromisso entre nível de segurança e custo (em tempo de processamento, por exemplo). Desta forma, é importante permitir a utilização apenas dos mecanismos de segurança necessários, evitando um desperdício de recursos. Isto se dá, por exemplo, pela possibilidade de seleção individual dos aspectos de segurança a necessários;
- Gradatividade de implantação: a adição gradual dos mecanismos de segurança nos membros da rede é também importante quando se imagina um cenário de atualização de um sistema como um todo. Soluções onde vários elementos devem ser atualizados em simultâneo demandam um esforço muito maior para serem implantadas, e sua utilização tende a ser postergada, muitas vezes, indefinidamente. Exemplos nesse sentido são comuns em redes de computadores. Tecnologias já bastante estudadas, como multicast ou IPv6, ainda têm bastante dificuldade para serem implantadas.

Em vista disso, é importante permitir uma integração natural entre *peers* que fazem uso da arquitetura de segurança e aqueles que não a utilizam;

- Re-configurabilidade: aplicações P2P tipicamente apresentam requisitos de segurança variáveis ao longo do tempo. No decorrer do uso de um sistema, algum novo aspecto de segurança pode se tornar necessário ou alguma restrição pode passar a não mais existir. Considerando esta característica, permitir a redefinição de requisitos de segurança sem necessidade de recompilação é um aspecto importante a ser atendido na definição da arquitetura.

4.2 Visão Geral

A idéia central da arquitetura proposta é a adição de uma camada de segurança às aplicações P2P. Esta camada é implementada e configurada de forma independente tanto da aplicação quanto do *middleware* de comunicação subjacente. Os aspectos de segurança são atendidos através de módulos, independentes entre si, que implementam diferentes técnicas de segurança. A definição e o ajuste dos módulos a serem empregados é feita de forma local e autônoma em cada um dos *peers*. A configuração da camada de segurança é baseada no estabelecimento de *perfis*, cada um abrangendo diferentes necessidades de segurança. Com isso, os *peers* identificados na rede podem ser agrupados em um dos perfis definidos. Para o caso de *peers* não conhecidos previamente, estabelece-se um perfil padrão a ser utilizado.

A Figura 4.1 ilustra um exemplo de instância dessa arquitetura. As configurações para o Peer 2 e o Peer 3 são mostradas em detalhes. O Peer 2 cria um perfil referenciando o uso de uma técnica de autenticação contendo o Peer 1 e o Peer 4. Autenticação é aplicada em todas as mensagens trocadas entre o Peer 2 e os Peers 1 e 4. O Peer 2 também especifica que autenticação deve ser aplicada quando do envio de mensagens para o Peer 3, enquanto autenticação e confidencialidade são exigidos quando do recebimento de mensagens deste *peer*. As configurações definidas no Peer 3 especificam que autenticação e confidencialidade são aplicadas quando do envio de mensagens para os Peers 1 e 2, mas apenas autenticação é exigida quando do recebimento de mensagens provenientes desses *peers*. Na configuração do Peer 3 em relação ao Peer 4, por sua vez, autenticação é exigida no envio e no recebimento, enquanto a geração de *logs* é aplicada apenas no recebimento de mensagens. As seções a seguir descrevem em mais detalhes os principais elementos da

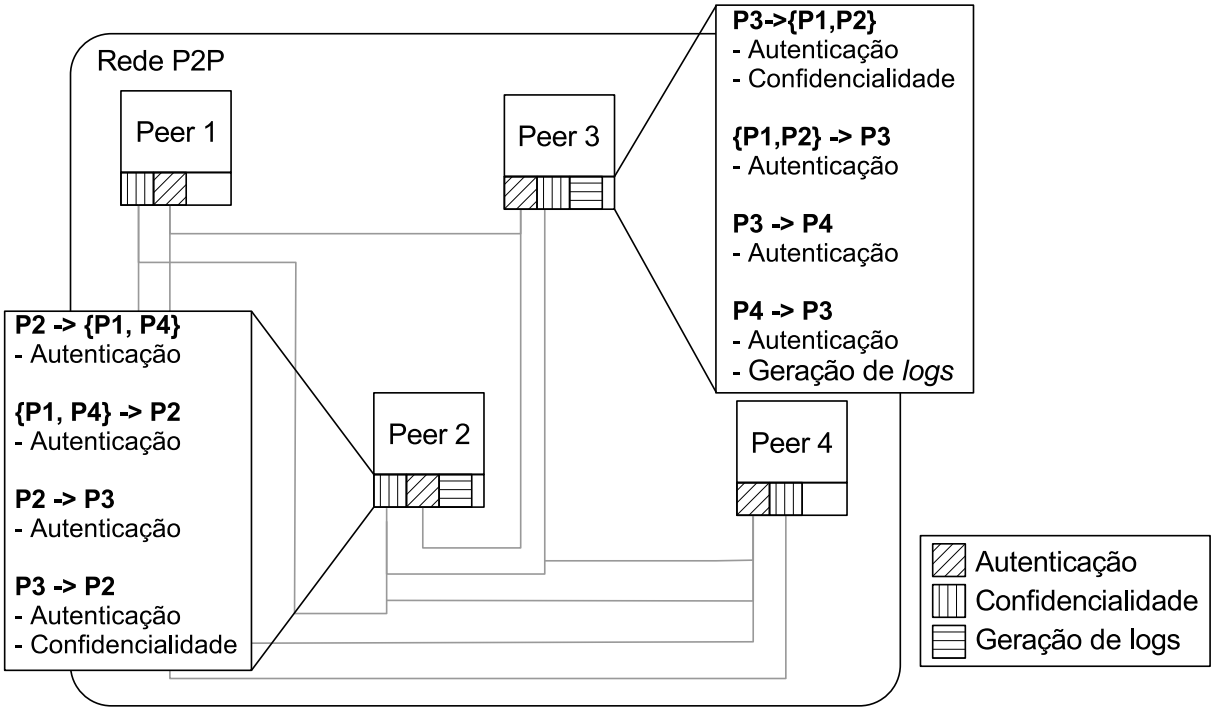


FIGURA 4.1 – Exemplo de instância da arquitetura.

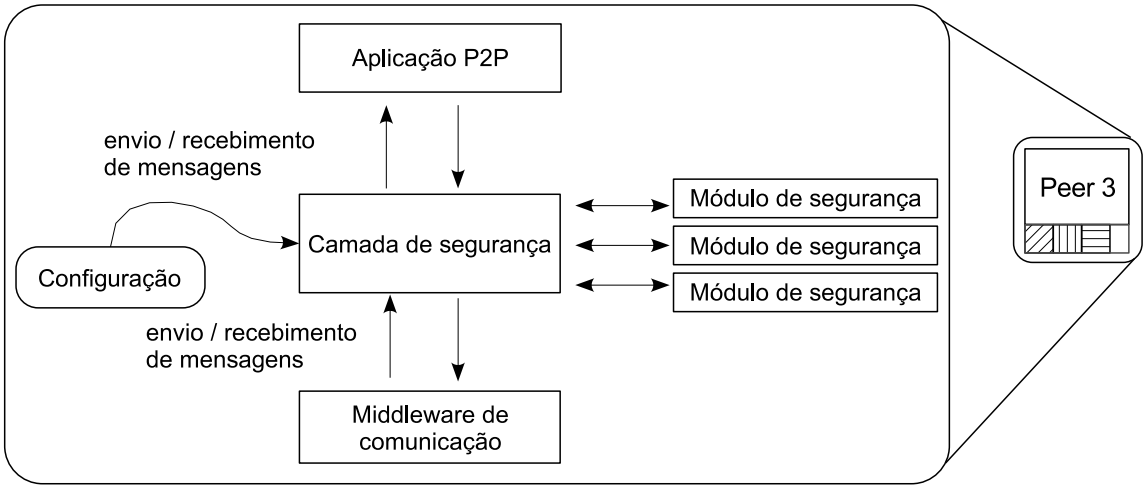


FIGURA 4.2 – Arquitetura da camada de segurança.

arquitetura.

4.3 Camada de segurança

A camada de segurança ligada ao *peer* durante a operação do sistema atua como um invólucro (*wrapper*) entre a aplicação P2P e o *middleware* de comunicação subjacente. Sempre que mensagens são recebidas ou enviadas, esta camada utiliza os módulos de segurança configurados de forma a garantir os aspectos de segurança estabelecidos. A Figura 4.2 ilustra este processo. Neste exemplo, sempre que uma mensagem chega ao Peer 3, ela passa pelos três módulos instanciados. Caso a passagem pelos módulos seja bem sucedida, a mensagem é entregue à aplicação. Caso contrário, dependendo das características do módulo, a mensagem é simplesmente descartada. Quando uma mensagem é enviada para outro *peer*, passa pelos módulos configurados antes de ser repassada ao *middleware* de comunicação.

4.4 Módulo de configuração

A configuração dos aspectos de segurança é realizada localmente no *peer*, mas de forma independente à aplicação. O processo de configuração é facilitado por um módulo que ingressa na rede P2P, buscando informações que guiem a escolha dos aspectos por

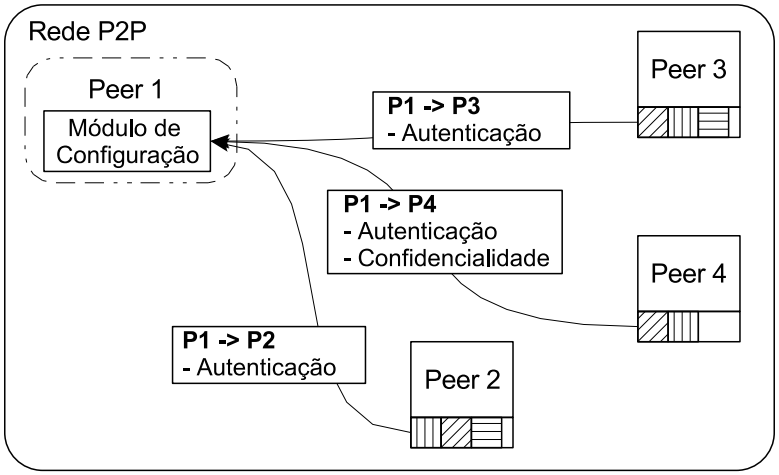


FIGURA 4.3 – Obtenção das configurações remotas.

parte do usuário. O módulo de configuração acessa a camada de segurança dos outros *peers* de modo a consultar quais aspectos de segurança estão sendo exigidos. Isto permite que a configuração local esteja de acordo com os aspectos exigidos nos demais *peers*. A Figura 4.3 ilustra o processo: o módulo de configuração do Peer 1 consulta os Peers 2, 3 e 4, que respondem com os requisitos referentes às mensagens a serem futuramente enviadas pelo Peer 1 a cada um deles. Além dos requisitos, os *peers* informam também quais módulos estão disponíveis para uso, bem como parâmetros relevantes aos *peers* remotos. Estas informações serão detalhadas nas Seções 4.6 e 4.7. Para evitar que as mensagens envolvidas nesta fase possam ser forjadas ou alteradas, pode ser empregado um esquema de troca segura de mensagens (como SSL), disponibilizado pelo *middleware* de comunicação ou implementado pela camada de segurança.

Realizada a descoberta dos *peers* e a consulta de seus requisitos e mecanismos de segurança empregados, o usuário pode decidir quais aspectos de segurança devem ser empregados localmente. Para tal, ele(a) especifica o conjunto de aspectos de segurança a ser observado em cada um dos canais de comunicação com os *peers*. Conforme citado no início desta seção, a especificação é facilitada através da criação de grupos de *peers* com características comuns, que estabeleçam um perfil de segurança. Terminada essa especificação, as configurações realizadas são armazenadas em um repositório local, que é acessado pela camada de segurança de forma a prover os aspectos especificados.

4.5 Módulos de segurança

Os módulos de segurança são as peças chave da arquitetura proposta. Cada técnica de segurança implementada é representada através de um desses módulos. Por exemplo, o suporte à auditoria da troca de mensagens é tipicamente implementado através de um módulo de geração de *logs*; a autenticação e a garantia de integridade das mensagens, através de um módulo de assinatura digital; e o *controle de acesso* aos recursos (autorização), através da verificação de políticas de acesso.

Os módulos atuam de forma independente entre si, e se baseiam na utilização de uma interface genérica de acesso que permite a extensão da arquitetura de forma simples. Cada módulo possui métodos para verificação de mensagens recebidas e para preparação de mensagens em processo de envio. Ao ser acionado pela camada de segurança, o módulo realiza o processamento necessário (podendo alterar o conteúdo da mensagem) e retorna se a operação foi bem sucedida ou não. Um exemplo disso é um módulo de assinatura digital, que adiciona uma assinatura aos dados da mensagem antes do seu envio e verifica a validade da mesma no recebimento da mensagem.

4.6 Repositório de caracterização dos módulos

Conceitualmente, os módulos de segurança diferem entre si em relação a parâmetros de entrada e mesmo à sua dinâmica de utilização. Enquanto técnicas como a verificação de autenticidade através de assinatura digital dependem que o remetente da mensagem aplique a assinatura para que o destinatário possa conferí-la, uma técnica de verificação de políticas de uso de recursos não precisa desta "cumplicidade" entre os *peers*.

Para lidar com essa heterogeneidade, evitando adicionar diretamente à arquitetura as características de cada módulo idealizado, foi definido um repositório independente contendo a caracterização dos módulos disponíveis. Este repositório é implementado através de um arquivo XML, que define os parâmetros e a dinâmica de uso de cada módulo, bem como as combinações de módulos que atendem a cada um dos aspectos de segurança oferecidos. A Figura 4.4 apresenta um exemplo de arquivo XML de caracterização, relativo aos módulos implementados nesta dissertação: `PgpEncryption`, `PgpSignature`, `Log` e `PoliciesChecking` (descritos individualmente adiante, na Seção 5.2). A seguir são explicados cada um dos principais elementos do repositório de caracterização, utilizando o arquivo da Figura 4.4 para exemplificar cada um deles.

```
<static_features>
  <modules>
    <module name="PgpEncryption"
            export_requirement="true"
            obligatory_if_applied="true"
            allow_on_bcast_sending="false"
            discard_on_failure="false" >
      <parameters>
        <parameter name="public_keys_file" default="~/.gnupg/pubring.gpg"/>
        <parameter name="secret_keys_file" default="~/.gnupg/secring.gpg"/>
        <parameter name="my_encryption_key_id" default="" remote="true"/>
        <parameter name="pass_phrase"      default=""/>
      </parameters>
    </module>

    <module name="PgpSignature"
            export_requirement="true"
            obligatory_if_applied="false"
            allow_on_bcast_sending="true"
            discard_on_failure="true" >
      <parameters>
        <parameter name="public_keys_file" default="~/.gnupg/pubring.gpg"/>
        <parameter name="secret_keys_file" default="~/.gnupg/secring.gpg"/>
        <parameter name="my_signing_key_id" default="" remote="true"/>
        <parameter name="pass_phrase"      default=""/>
      </parameters>
    </module>

    <module name="Log"
            export_requirement="false"
            obligatory_if_applied="false"
            allow_on_bcast_sending="true"
            discard_on_failure="false" >
      <parameters>
        <parameter name="log_level"  default="1" options="low medium high"/>
        <parameter name="extra_fields"  default=""/>
        <parameter name="output_file"   default=""/>
      </parameters>
    </module>

    <module name="PoliciesChecking"
            export_requirement="false"
            obligatory_if_applied="false"
            allow_on_bcast_sending="true"
            discard_on_failure="true" >
      <parameters>
        <parameter name="policies_file"  default=""/>
      </parameters>
    </module>
  </modules>
```

```
51
52    <requirements>
53      <requirement name="Confidentiality">
54        <option>
55          <option_module name="PgpEncryption"/>
56        </option>
57      </requirement>
58
59      <requirement name="Authentication">
60        <option>
61          <option_module name="PgpSignature"/>
62        </option>
63      </requirement>
64
65      <requirement name="Integrity">
66        <option>
67          <option_module name="PgpSignature"/>
68        </option>
69      </requirement>
70
71      <requirement name="Authorization">
72        <option>
73          <option_module name="PoliciesChecking"/>
74        </option>
75      </requirement>
76
77      <requirement name="Auditing">
78        <option>
79          <option_module name="Log"/>
80        </option>
81      </requirement>
82    </requirements>
83  </static_features>
```

FIGURA 4.4 – Exemplo de caracterização de módulos.

Os *parâmetros* dos módulos se referem àquelas informações que o administrador do *peer* deve ajustar para que o módulo possa funcionar adequadamente. O arquivo da Figura 4.4 apresenta vários exemplos de parâmetros, dentre os quais: a identificação de uma chave a ser utilizada na cifragem ou assinatura digital (linhas 11 e 24); o nível de detalhamento a ser obedecido por um módulo de geração de *logs* (linha 35)*;* e o caminho para algum arquivo de especificação de políticas de acesso (linha 47), no caso do módulo responsável pela garantia de autorização. Alguns parâmetros especificados localmente podem, adicionalmente, ser importantes para outros *peers* na rede. Por exemplo, se um

peer deseja cifrar uma mensagem utilizando criptografia assimétrica, ele deve saber qual a chave pertence ao *peer* destino, de forma que a operação possa ser realizada com sucesso. Para esses casos, especifica-se um atributo (`remote`) que indica que este parâmetro deve ser informado aos outros *peers* no caso de uma consulta por requisitos (conforme foi explicado na Seção 4.4). O arquivo de caracterização apresentado na Figura 4.4 possui dois exemplos de parâmetros com o atributo `remote` presente, relativos à identificação da chave de cifragem usada no módulo de cifragem (linha 11) e identificação da chave empregada no módulo de assinatura digital (linha 24).

A *dinâmica de uso* de um módulo diz respeito às possibilidades e implicações da sua utilização. Em relação a isso, são definidos quatro atributos para um módulo:

- `export_requirement`: define se o módulo deve ser empregado pelo transmissor para que possa ser utilizado pelo destinarário. Por exemplo, para que uma mensagem possa ter a sua autenticidade verificada no recebimento, o transmissor precisa inserir um verificador no momento do envio. Por isso, módulos ligados à autenticação de mensagens, como é o caso do módulo `PgpSignature` caracterizado na Figura 4.4 (linha 17), possuem esse atributo ajustado para verdadeiro;

- `obligatory_if_applied`: indica se as mensagens devem ser repassadas ao módulo no recebimento sempre que este foi aplicado no envio. Por exemplo, uma mensagem alterada por algum módulo de cifragem durante o envio deve necessariamente ser repassada para o mesmo módulo no recebimento, para que os dados possam ser acessados. Este é o caso do módulo `PgpEncryption`, na qual o atributo `obligatory_if_applied` é definido como verdadeiro na linha 5 da Figura 4.4;

- `allow_on_bcast_sending`: especifica se o módulo pode ser aplicado durante o envio de mensagens via *broadcast*;

- `discard_on_failure`: define se uma mensagem recebida deve ou não ser descartada quando falha ao passar pelo módulo.

O repositório também armazena o mapeamento entre módulos e aspectos de segurança. Note-se que um módulo pode atender a mais de um aspecto de segurança, assim como um aspecto pode demandar o uso conjunto de múltiplos módulos. Por exemplo, a combinação de um módulo para a geração de um verificador via função *hash* MD5 com outro para assinatura deste verificador representa uma opção para autenticação de mensagens. A

TABELA 4.1 – Mapeamento entre aspectos de segurança e técnicas utilizadas.

Aspecto	Técnicas
Confidencialidade	Criptografia PGP Criptografia RSA Criptografia RC4
Autenticidade	Assinatura PGP Assinatura RSA Criptografia RC4 Código de Autenticação de Mensagem Função SHA1 + Assinatura PGP sobre verificador Função SHA1 + Assinatura RSA sobre verificador Função SHA1 + Assinatura RC4 sobre verificador
Integridade	Assinatura PGP Assinatura RSA Criptografia RC4 Código de Autenticação de Mensagem Função SHA1 + Assinatura PGP sobre verificador Função SHA1 + Assinatura RSA sobre verificador Função SHA1 + Assinatura RC4 sobre verificador
Autorização	Verificação de políticas de acesso
Auditoria	Geração de *log*

Tabela 4.1 ilustra o mapeamento entre aspectos de segurança e exemplos de técnicas que podem ser empregadas na arquitetura proposta. Note-se que o exemplo da Figura 4.4, que mapeia os aspectos de segurança atendidos pelos módulos nas linhas de 52 a 82, oferece apenas uma opção (`<option>`) para atender a cada um dos aspectos de segrança definidos. Entretanto, já se planeja a adição de outras opções de mecanismos, como aqueles apresentados na Tabela 4.1.

O arquivo XML de caracterização dos módulos pode ser editado livremente de modo a contemplar novos módulos ou novas características. A DTD (*Document Type Definitions*) que rege o formato deste arquivo é apresentada na Figura 4.5.

```
1  <!ELEMENT static_features (modules?) (requirements?)>
2  <!ELEMENT modules (module*)>
3  <!ELEMENT module (paramaters?)>
4  <!ATTLIST module name CDATA #REQUIRED>
5  <!ATTLIST module export_requirement     ("true"|"false") "false">
6  <!ATTLIST module obligatory_if_applied  ("true"|"false") "false">
7  <!ATTLIST module allow_on_bcast_sending ("true"|"false") "false">
8  <!ATTLIST module discard_on_failure     ("true"|"false") "false">
9  <!ELEMENT parameters (parameter*)>
10 <!ELEMENT parameter EMPTY>
11 <!ATTLIST parameter name CDATA #REQUIRED>
12 <!ATTLIST parameter default CDATA #REQUIRED>
13 <!ATTLIST parameter options CDATA #IMPLIED>
14 <!ELEMENT requirements (requirement*)>
15 <!ELEMENT requirement (option*)>
16 <!ATTLIST requirement name CDATA #REQUIRED>
17 <!ELEMENT option (option_module+)>
18 <!ATTLIST option_module name CDATA #REQUIRED>
```

FIGURA 4.5 – DTD referente ao XML de caracterização dos módulos.

4.7 Repositório de configuração

O repositório de configuração contém as informações necessárias para que a camada de segurança aplique adequadamente os módulos especificados pelo administrador. O repositório é implementado na forma de um arquivo XML, exeplificado na Figura 4.6. Este exemplo condiz às configurações do Peer 3 no cenário apresentado na Figura 4.1 (Seção 4.2). O arquivo XML está dividido em três partes: `peers`, `profiles` e `available_modules`, conforme explicado a seguir.

```
<?xml version="1.0" ?>
<settings>
  <peers>
    <peer name="Peer1">
      <available_modules>
        <module name="PgpEncryption">
          <parameter name="my_encryption_key" value="739d7f591fc40a56"/>
        </module>
        <module name="PgpSignature">
          <parameter name="my_signing_key"    value="7199103DADF135ED"/>
        </module>
        <module name="Log"/>
        <module name="PoliciesChecking"/>
      </available_modules>
      <required_modules>
        <module name="PgpSignature">
          <parameter name="my_signing_key"    value="7199103DADF135ED"/>
        </module>
      </required_modules>
    </peer>
    <peer name="Peer2">
      <available_modules>
        <module name="PgpEncryption">
          <parameter name="my_encryption_key" value="739d7f591fc40a57"/>
        </module>
        <module name="PgpSignature">
          <parameter name="my_signing_key"    value="7199103DADF135EE"/>
        </module>
        <module name="Log"/>
        <module name="PoliciesChecking"/>
      </available_modules>
      <required_modules>
        <module name="PgpEncryption">
          <parameter name="my_encryption_key" value="739d7f591fc40a57"/>
        </module>
        <module name="PgpSignature">
          <parameter name="my_signing_key"    value="7199103DADF135EE"/>
        </module>
      </required_modules>
    </peer>
    <peer name="Peer4">
      <available_modules>
        <module name="PgpEncryption">
          <parameter name="my_encryption_key" value="739d7f591fc40a58"/>
        </module>
        <module name="PgpSignature">
          <parameter name="my_signing_key"    value="7199103DADF135EF"/>
        </module>
        <module name="Log"/>
        <module name="PoliciesChecking"/>
```

```
51        </available_modules>
52        <required_modules>
53          <module name="PgpEncryption">
54            <parameter name="my_encryption_key" value="739d7f591fc40a58"/>
55          </module>
56          <module name="PgpSignature">
57            <parameter name="my_signing_key"    value="7199103DADF135EF"/>
58          </module>
59        </required_modules>
60      </peer>
61    </peers>
62
63    <profiles>
64      <profile name="[default]" respect_remote_requirements="true">
65        <incoming_requirements>
66          <requirement name="Confidentiality"/>
67          <requirement name="Authentication"/>
68          <requirement name="Authorization"/>
69        </incoming_requirements>
70        <outgoing_requirements>
71          <requirement name="Authentication"/>
72        </outgoing_requirements>
73        <incoming_modules>
74          <module name="PgpEncryption"/>
75          <module name="PgpSignature"/>
76          <module name="PoliciesChecking"/>
77          <module name="Log">
78            <parameter name="log_level"         value="high"/>
79            <parameter name="extra_fields"      value=""/>
80            <parameter name="output_file"       value="~/log.out"/>
81          </module>
82        </incoming_modules>
83        <outgoing_modules>
84          <module name="PgpSignature"/>
85        </outgoing_modules>
86      </profile>
87      <profile name="[broadcast]" respect_remote_requirements="true">
88        <incoming_modules/>
89        <incoming_requirements/>
90        <outgoing_requirements>
91          <requirement name="Authentication"/>
92        </outgoing_requirements>
93        <outgoing_modules>
94          <module name="PgpSignature"/>
95        </outgoing_modules>
96      </profile>
97      <profile name="ProfileA" respect_remote_requirements="true">
98        <incoming_requirements>
99          <requirement name="Authentication"/>
100       </incoming_requirements>
101       <outgoing_requirements>
```

```
102         <requirement name="Confidentiality"/>
103         <requirement name="Authentication"/>
104       </outgoing_requirements>
105       <incoming_modules>
106         <module name="PgpSignature"/>
107       </incoming_modules>
108       <outgoing_modules>
109         <module name="PgpEncryption"/>
110         <module name="PgpSignature"/>
111       </outgoing_modules>
112       <peers_from>
113         <peer_from name="Peer1"/>
114         <peer_from name="Peer2"/>
115       </peers_from>
116     </profile>
117     <profile name="ProfileB" respect_remote_requirements="true">
118       <incoming_requirements>
119         <requirement name="Authentication"/>
120         <requirement name="Auditing"/>
121       </incoming_requirements>
122       <outgoing_requirements>
123         <requirement name="Authentication"/>
124       </outgoing_requirements>
125       <incoming_modules>
126         <module name="PgpSignature"/>
127         <module name="Log"/>
128       </incoming_modules>
129       <outgoing_modules>
130         <module name="PgpSignature"/>
131       </outgoing_modules>
132       <peers_from>
133         <peer_from name="Peer4"/>
134       </peers_from>
135     </profile>
136   </profiles>
137
138   <available_modules>
139     <module name="PgpEncryption">
140       <parameter name="pass_phrase"       value="XXXXXXXXXXX"/>
141       <parameter name="public_keys_file"  value="~/.gnupg/pubring.gpg"/>
142       <parameter name="secret_keys_file"  value="~/.gnupg/secring.gpg"/>
143       <parameter name="my_encryption_key" value="739d7f591fc40a55"/>
144     </module>
145     <module name="PgpSignature">
146       <parameter name="pass_phrase"       value="XXXXXXXXXXX"/>
147       <parameter name="public_keys_file"  value="~/.gnupg/pubring.gpg"/>
148       <parameter name="secret_keys_file"  value="~/.gnupg/secring.gpg"/>
149       <parameter name="my_signing_key"    value="7199103DADF135EC"/>
150     </module>
151     <module name="Log">
152       <parameter name="log_level"         value="medium"/>
```

```
153       <parameter name="extra_fields"      value=""/>
154       <parameter name="output_file"       value="~/log.out"/>
155     </module>
156     <module name="PoliciesChecking">
157       <parameter name="peers_roles"       value="~/peers_roles.xml"/>
158       <parameter name="policies_file"     value="~/policies.xml"/>
159     </module>
160   </available_modules>
161 </settings>
```

FIGURA 4.6 – Exemplo de configuração da arquitetura de segurança.

Na seção `peers` (linhas 3 a 61 da Figura 4.6) estão listados os *peers* remotos identificados na rede, bem como as suas características coletadas através do módulo de configuração: módulos disponíveis e módulos exigidos no envio de mensagens do *peer* local para o *peer* remoto. No exemplo apresentado, pode-se identificar os módulos disponíveis no Peer1 (linhas 5 a 14) e observar que este *peer* exige a aplicação do módulo `PgpSignature` no caso de mensagens enviadas pelo Peer3 (linhas 15 a 19). Além de servir de referência para o administrador do *peer* sobre quais módulos podem ser empregados no envio de mensagens, a listagem de módulos disponíveis remotamente é importante para o caso de módulos com parâmetros onde o atributo `remote` é verdadeiro. Através da coleta desse tipo de informação, o *peer* local pode, por exemplo, cifrar adequadamente uma mensagem, utilizando uma chave pública designada pelo *peer* remoto (como na linha 10, referente ao Peer1). Já a informação de quais módulos são exigidos no recebimento de mensagens por parte do *peer* remoto é relevante para que o *peer* local possa se adequar a estes requisitos no momento do envio de mensagens.

A seção `profiles` (linhas 63 a 136) contém a especificação dos perfis na qual podem ser enquadrados os *peers* que compõem a rede P2P. Além dos perfis definidos pelo administrador, a arquitetura prevê a existência de dois perfis especiais, os quais não possuem *peers* específicos associados: `[default]` (linha 64) e `[broadcast]` (linha 87). As configurações definidas no perfil `[default]` são tomadas como referência quando o *peer* de destino ou de origem da mensagem não está contido em nenhum outro perfil. Já o perfil `[broadcast]` é empregado quando uma mensagem vai ser enviada por *broadcast*.

Fazem parte da configuração de cada perfil os requisitos desejados na chegada e na saída de mensagens (`incoming_requirements` e `outgoing_requirements`) e os módulos escolhidos para satisfazer esses requisitos (`incoming_modules` e `outgoing_modules`). Quando da escolha dos módulos a serem aplicados, o administrador pode definir parâme-

tros específicos para os módulos no contexto de cada perfil. Isto permite definir diferentes chaves a serem empregadas na assinatura ou diferentes níveis de detalhamento de *log*, de acordo com os *peers* que compõem cada perfil. É o caso do módulo de geração de *logs* no perfil `[default]` (linhas 77 a 81), onde é empregado um nível de detalhamento maior que o utilizado nas outras situações. Na configuração de um perfil o administrador tem a opção de definir, através do atributo `respect_remote_requirements` (linhas 97 e 117) se os requisitos dos *peers* remotos pertencentes a este perfil devem ser obedecidos mesmo se a configuração do perfil não prevê o uso dos módulos exigidos. Sempre que uma mensagem estiver sendo enviada para um *peer* de um perfil com este atributo ativado, a mensagem passa tanto pelos módulos previstos no perfil quanto pelos módulos exigidos pelo *peer* destinatário. A utilização deste atributo permite que a heterogeneidade de requisitos seja tratada de forma mais automática e simples.

Por fim, a seção `available_modules` define os módulos disponíveis localmente, bem como os valores padrão dos parâmetros (adotados quando não especificados dentro dos perfis). Os módulos listados nessa seção, bem como os valores especificados para os parâmetros definidos como `remote` no arquivo de caracterização (Seção 4.6) são enviados para os *peers* remotos que requisitam as informações do *peer* local.

A DTD que define o formato do arquivo XML de configuração é apresentada na Figura 4.7.

```
<!ELEMENT settings (peers?) (profiles?) (available_modules?)>
<!ELEMENT peers (peer*)>
<!ELEMENT peer (available_modules?) (required_modules?)>
<!ELEMENT available_modules (module*)>
<!ELEMENT required_modules (module*)>
<!ELEMENT module (parameter*)>
<!ATTLIST module name CDATA #REQUIRED>
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter name CDATA #REQUIRED>
<!ATTLIST parameter value CDATA #REQUIRED>
<!ELEMENT profiles (profile*)>
<!ELEMENT profile (peers_from?)
                  (incomming_requirements?) (outgoing_requirements?)
                  (incomming_modules?) (outgoing_modules?) >
<!ELEMENT incoming_requirements (requirement*)>
<!ELEMENT outgoing_requirements (requirement*)>
<!ELEMENT requirement EMPTY>
<!ATTLIST requirement name CDATA #REQUIRED>
<!ELEMENT incoming_modules (module*)>
<!ELEMENT outgoing_modules (module*)>
<!ELEMENT peers_from (peer_from*)>
<!ATTLIST peers_from name CDATA #REQUIRED>
```

FIGURA 4.7 – DTD referente ao XML de configuração dos módulos.

Capítulo 5

Implementação

A arquitetura de segurança descrita na seção anterior foi implementada em Java, utilizando JXTA ([8], [62]) como *middleware* de comunicação subjacente. O JXTA é um projeto que visa estabelecer um conjunto de protocolos, independentes de implementação, que possibilitam a criação de uma estrutura P2P de uso geral, independente de aplicação. Mais especificamente, a implementação foi baseada na JXTA Abstraction Layer (JAL - [66]), uma biblioteca que tem por objetivo principal facilitar o desenvolvimento de aplicações P2P sobre JXTA. JAL abstrai vários aspectos da arquitetura do JXTA, oferecendo ao programador uma interface mais simples para acesso a funcionalidades comuns em sistemas P2P, como envio de mensagens (*unicast* ou *broadcast*), criação de grupos e localização de recursos. JXTA é amplamente adotado em uma grande variedade de projetos e trabalhos científicos, de forma que a implementação utilizada pode ser empregada em vários sistemas já existentes. O modelo da arquitetura, bem como boa parte da implementação, se adequam bem a outros *middlewares* de comunicação. É exigido, entretanto, que a comunicação entre os *peers* seja realizada obedecendo uma semântica de envio/recebimento de mensagens. Adicionalmente, para que seja possível manter a possibilidade de comunicação com entidades que não implementem a camada de segurança, o *middleware* deve permitir que campos sejam adicionados às mensagens sem comprometimento do acesso aos dados já presentes.

5.1 Camada de segurança

A Figura 5.1 ilustra a implementação da camada de segurança definida na Seção 5.1. A principal peça de implementação é a classe `SecurePeer`, que atua como o in-

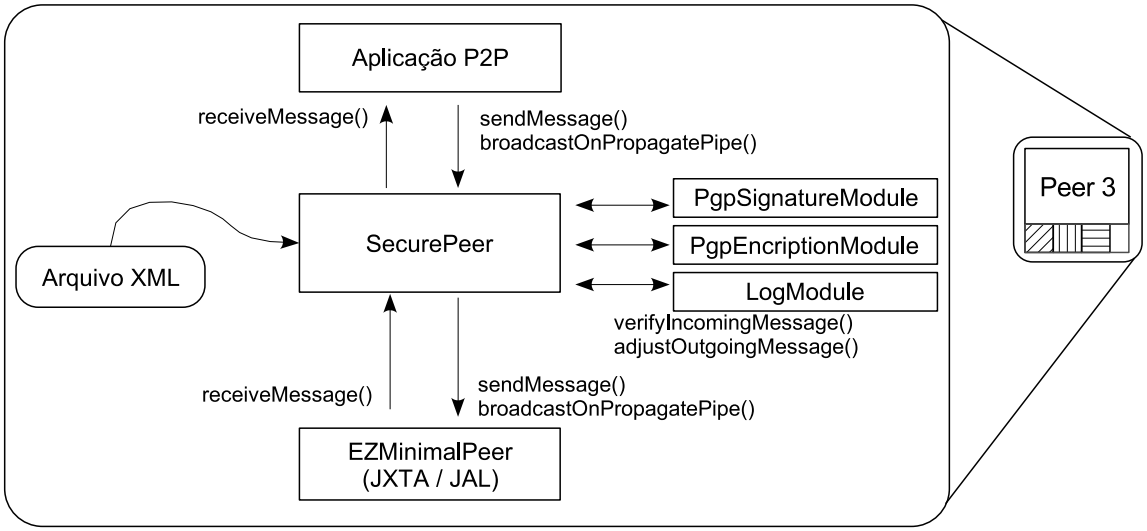


FIGURA 5.1 – Implementação da camada de segurança.

vólucro que intercepta mensagens sendo enviadas ou recebidas pelo *peer*. Os principais métodos disponibilizados são: inicialização (método `boot`), onde os módulos são instanciados e configurados, envio por *broadcast* (`broadcastOnPropagatePipe`), envio por *unicast* (`sendMessage`) e recebimento de mensagens (`receiveMessage`). A utilização da camada do ponto de vista da aplicação baseada em JXTA/JAL implica a substituição do uso da classe `EZMinimalPeer` (disponibilizada pelo JAL) pelo uso da classe `SecurePeer` (que por sua vez estende `EZMinimalPeer`).

Os módulos são especializações da classe `SecurityModule`, que oferece uma interface genérica de acesso, utilizada pela classe `SecurePeer`. Os métodos previstos em `SecurityModule` representam a verificação de cada mensagem recebida (`verifyIncomingMessage`) e a adequação de uma mensagem em processo de envio aos requisitos especificados (`adjustOutgoingMessage`). O acesso aos módulos é feito unicamente através desta interface genérica, o que, aliado ao uso de carga dinâmica de classes do Java na instanciação de módulos, torna a camada de segurança *extensível*: novos módulos podem ser adicionados sem a necessidade de recompilação do restante da camada.

A seguir, explica-se em linhas gerais os algoritmos aplicados no recebimento e envio de mensagens. Sempre que uma mensagem deve ser enviada, a instância de `SecurePeer` verifica as configurações do perfil relacionado ao *peer* destino. Caso o atributo `obey_remote_requirements` esteja ativado, a mensagem é primeiramente repassada para

os módulos de segurança exigidos pelo *peer* remoto (informação coletada pelo módulo de configuração). Em seguida, são acessados os módulos escolhidos localmente, relacionados ao perfil em questão. Sempre que a mensagem passa por um módulo que possui o atributo `export_requirement` ativado, o nome do módulo é adicionado a uma lista, que por sua vez é inserida na mensagem na forma de um campo denominado `APPLIED_MODULES`.

No recebimento da mensagem no *peer* destino, a lista inserida no campo `APPLIED_MODULES` é utilizada como indicação de quais módulos que exigem cooperação do remetente (atributo `export_requirement` ativado) *podem* ser utilizados no recebimento. Por exemplo, a inserção de uma assinatura digital no momento do envio permite ao destinarário verificar esta assinatura, que do contrário não poderia fazê-lo. Cada módulo dessa lista é acessado (método `verifyIncomingMessage`) se o mesmo é exigido localmente (consultando as configurações do perfil relativo ao *peer* de origem) ou se o módulo possui o atributo `obligatory_if_applied` ativado. Neste último caso, o módulo é empregado mesmo que não exigido na configuração local já que este atributo define que, uma vez empregado no envio, o módulo deve também ser utilizado no recebimento, como no caso de um módulo de criptografia. Uma vez tendo passado pelos módulos que possuem o atributo `export_requirement` ativado, a mensagem é repassada para os módulos que não possuam esta dependência, como no caso da verificação de políticas ou geração de *logs*.

O método `verifyIncomingMessage` dos módulos retorna um valor booleano que indica se a passagem da mensagem pelo módulo foi bem sucedida. Por exemplo, um módulo de criptografia retorna se a mensagem estava corretamente cifrada; um módulo de assinatura retorna se a assinatura é legítima; e um módulo de verificação de políticas retorna se a mensagem está autorizada a ser entregue. Caso um módulo retorne falso e o módulo tenha na sua caracterização o atributo `discard_on_failure` ativado, a mensagem é descartada. O mesmo acontece no caso de algum módulo que exige cooperação do *peer* remoto (atributo `export_requirement` ativado) não ter sido aplicado, apesar de ser exigido localmente para o canal de comunicação relativo à mensagem recebida.

5.2 Módulos implementados

Foram implementados módulos que atendem aos aspectos de segurança mais relevantes às aplicações P2P em redes corporativas: autenticação, confidencialidade, integridade, autorização e possibilidade de auditar a troca de mensagens, conforme apontado na Seção 2.3. Além destes aspectos já contemplados, é possível conceber módulos adequáveis à

arquitetura que possam prover reputação e não-repúdio. Para o caso de reputação, um módulo pode consultar campos configurados pelo usuário do *peer* de forma a identificar quais mensagens representam aumento de reputação e quais representam decréscimo na reputação do *peer* remoto. Como forma de prover não-repúdio, um módulo de autenticação pode ser especializado de modo a registrar as mensagens relevantes em um servidor confiável (*trusted*). Por outro lado, a implementação de anonimicidade exige a instanciação de mecanismos mais complexos, não podendo ser contemplada pela simples adição de módulos à arquitetura.

A implementação da arquitetura foi realizada de modo que novos módulos podem ser adicionados sem a necessidade de alteração ou recompilação. Os módulos podem ainda ser especializados para alguma aplicação, por exemplo utilizando campos específicos presentes nas mensagens como forma de autorizar ou não a sua entrega. Os módulos foram implementados de forma a permitir que *peers* que não implementam a camada de segurança possam, sempre que possível e não especificado de forma contrária na configuração dos outros *peers*, continuar participando no sistema. A seguir, são apresentados os módulos implementados. O arquivo de caracterização utilizado como exemplo na Seção 4.6 (Figura 4.4) apresenta características específicas de cada um dos módulos, podendo ser utilizado como referência adicional a esta seção.

5.2.1 Assinatura PGP

O módulo de assinatura PGP tem por objetivo assegurar a autenticidade e a integridade das mensagens trocadas entre os *peers*. É empregado o modelo de troca de chaves descentralizado, característico de PGP (*Pretty Good Privacy* - [69]), que dispensa a existência de uma autoridade de certificação central. Nesse modelo, as chaves públicas são armazenadas em servidores quaisquer, ou mesmo trocadas diretamente entre os interessados. A validade das chaves é assegurada através de uma "teia de confiança" formada pelos usuários de chaves PGP, conforme já mencionado na Seção 2.2. Ferramentas como o *Gnupg* ou *Kgpg* acessam servidores abertos ou específicos que contenham as chaves públicas de outros usuários e permitem que essas chaves sejam copiadas localmente para utilização futura. Essas ferramentas gerenciam também a criação de pares de chaves e a devida publicação das chaves públicas geradas. PGP permite o uso de diversos algoritmos de geração de assinatura, como RSA ou DSA, bem como a especificação do tamanho de chaves a ser empregado. Ambas as características são especificadas no momento da criação do par de chaves.

O módulo implementado utiliza as chaves públicas e privadas existentes localmente na estação onde o *peer* é executado. A chave privada utilizada é estabelecida previamente pelo administrador do *peer* (através de um parâmetro do módulo).

A geração da assinatura é precedida da criação de uma tabela *hash* temporária onde são armazenados todos os campos de dados da mensagem a ser enviada. A tabela criada é então convertida em um vetor de bytes, através do processo de *serialização* disponibilizado pelo Java. Esse vetor é passado para a algoritmo de geração de assinaturas apropriado, tarefa para qual é empregada a biblioteca *BouncyCastle* ([70]). Nesse passo, é gerada a assinatura, que consiste em um vetor de bytes com tamanho fixo, independente do tamanho do vetor original. Essa assinatura é adicionada à mensagem na forma de um campo `PGP_SIGNATURE_BYTES`. No recebimento da mensagem, um processo similar é realizado: uma tabela *hash* equivalente à preparada no envio é criada, serializada, e os bytes resultantes são verificados em relação à assinatura recebida, com base na chave pública do transmissor. Caso a assinatura seja válida, a mensagem pode ser repassada a outros módulos ou entregue à aplicação. Caso contrário, e se o *peer* foi configurado de forma a exigir a autenticação, a mensagem é descartada.

Conforme pode ser observado na Figura 4.4 (linhas 21 a 26), além da identificação (hexadecimal) do par de chaves cuja chave privada é empregada na assinatura, são parâmetros de entrada deste módulo os caminhos para os arquivos de chaves públicas e chaves privadas armazenados localmente, e a *pass phrase*, espécie de senha sem a qual a chave privada não pode ser utilizada.

5.2.2 Criptografia PGP

Assim como para o caso de geração de assinaturas, foi implementado um módulo que utiliza as facilidades oferecidas pelo PGP para garantir a confidencialidade das mensagens. Também no caso da criptografia, é empregado o modelo descentralizado de distribuição de chaves, especialmente interessante para aplicações P2P.

O módulo de criptografia PGP implementado retira da mensagem em processo de envio todos os campos de dados. Estes campos são colocados em uma tabela *hash* que é serializada em um vetor de bytes. Este vetor de bytes gerado é criptografado através das rotinas de cifragem PGP oferecidas pela biblioteca *BouncyCastle* e inserido na mensagem (substituindo os campos originais), em um campo `PGP_ENCRYPTED_BYTES`. No recebimento, este vetor de bytes é decifrado, e os campos de dados originais são reconstruídos e reinseridos na mensagem, de forma que a aplicação possa acessá-los normalmente.

São parâmetros de entrada deste módulo os caminhos para os arquivos de chaves públicas e chaves privadas armazenados localmente (tipicamente, os mesmos caminhos fornecidos para o módulo de assinatura PGP), a identificação (hexadecimal) do par de chaves cuja chave pública deve ser empregada pelos outros *peers* na cifragem das mensagens e a *pass phrase* que permite o uso da chave privada na decifragem das mensagens.

5.2.3 Verificação de políticas de acesso

Visando fornecer controle de acesso aos recursos (autorização), o módulo de verificação de políticas utiliza informações genéricas da comunicação (como data, tamanho das mensagens e remetente) para definir se a mensagem pode ou não ser entregue. Para a especificação e verificação das políticas é utilizado XACML ([71]), um padrão criado pelo OASIS para definição de políticas de controle de acesso através de XML. As políticas são definidas enquadrando os *peers* da rede em papéis, seguindo o modelo *Role Based Access Control* (RBAC). Com isso, são estabelecidos dois repositórios distintos (ambos implementados através de arquivos XML): um para enquadrar os participantes nos papéis definidos e outro para a especificação das políticas de acesso. O arquivo que mapeia os *peers* aos papéis correspondentes é exemplificado na Figura 5.2, que ilustra um possível mapeamento realizado pelo administrador do Peer 3, em uma rede com quatro *peers*. Os papéis definidos são "Universidades" e "Empresas". Note-se que *peers* podem pertencer a mais de um papel, como no caso do Peer 4.

A Figura 5.3 apresenta uma política de acesso no formato XACML. Nas linhas 10 e 11, dentro da seção de sujeitos (`<subjects>`) da política, é especificado o papel (*role*) a qual a política se aplica, no caso, "Universidades". A partir da linha 18 é especificada a primeira regra (`<Rule>`) referente à política descrita. O atributo `Effect` define o efeito da regra, que pode ser uma permissão de acesso (`Permit`, como no exemplo da figura) ou uma negação (`Deny`). Uma regra possui duas áreas principais: `<Target>` e `<Condition>`. Basicamente, `<Target>` define as características das mensagens que se aplicam a essa regra. `<Target>` considera sujeitos (`<Subjects>`), recursos (`<Resources>`) e ações (`<Actions>`). A regra definida no exemplo tem o seu `<Target>` descrito apenas a partir de recursos (uma vez que os sujeitos já foram definidos no nível acima, da política). A *tag* `<Resources>`, por sua vez, pode ser composta de várias ocorrências de `<Resource>`. No exemplo, é utilizada apenas uma dessas ocorrências, que restringe o tamanho das mensagens que podem ser recebidas. Primeiramente (linhas 23 a 25) o atributo é designado com o seu nome (`MSG:Size`) e tipo (inteiro). Em seguida (linhas 28 a 31), especifica-se que o va-

lor `20.000` deve ser maior ou igual (`integer-greater-than-or-equal`) ao tamanho, em bytes, da mensagem recebida. Retomando as outras áreas de uma regra, `<Condition>` descreve a condição que resultará na decisão de acesso. Se a condição retornar verdadeiro, o efeito da regra (`Permit` ou `Deny`) é retornado como resposta à requisição. No módulo aqui descrito, a área `<Condition>` é utilizada para as decisões sobre os períodos do dia em que as requisições devem ser autorizadas. Para isso, é utilizada a expressão especial `current-time`, que é comparada aos horários definidos em `<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">`, nas linhas 45 a 50. Ainda na mesma política é definida outra regra, com efeito de negação de acesso (`Deny`), para o caso de mensagens relativas ao papel "Universidades" que não atendam às características definidas na seção `target` da regra anterior (linha 56).

Sempre que uma mensagem é recebida, identifica-se os papéis que representam o remetente e prepara-se uma *string* de consulta também em formato XML, com base nas informações relevantes ao controle de acesso. A biblioteca Sun'S XACML ([72]), empregada na implementação do módulo, processa a string e, com base no repositório de políticas, retorna se o acesso deve ou não ser permitido. Neste último caso, a mensagem é descartada, sem ser entregue à aplicação.

```
1  <?xml version='1.0' encoding='utf-8'?>
2  <peerRepository>
3    <peer>
4      <name>Peer1</name>
5      <role>Universidades</role>
6    </peer>
7    <peer>
8      <name>Peer2</name>
9      <role>Empresas</role>
10   </peer>
11   <peer>
12     <name>Peer4</name>
13     <role>Universidades</role>
14     <role>Empresas</role>
15   </peer>
16 </peerRepository>
```

FIGURA 5.2 – Exemplo de mapeamento entre *peers* e papéis.

```
<Policy PolicyId="Policy:to:Universidades:role" RuleCombiningAlgId=
"urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
  <Target>
    <Subjects>
      <Subject>
        <SubjectMatch MatchId=
        "urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <SubjectAttributeDesignator AttributeId="urn:someapp:attributes:
          role" DataType="http://www.w3.org/2001/XMLSchema#string"/>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          Universidades</AttributeValue>
        </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources> <AnyResource/> </Resources>
    <Actions>   <AnyAction/>   </Actions>
  </Target>
  <Rule RuleId="Rule:to:Universidades:role" Effect="Permit">
    <Target>
      <Subjects> <AnySubject/> </Subjects>
      <Resources>
        <Resource>
            <ResourceAttributeDesignator
            AttributeId="MSG:Size" DataType="http://www.w3.org/2001/
            XMLSchema#integer"/>
          <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:
          function:integer-greater-than-or-equal">
            <AttributeValue DataType="http://www.w3.org/2001/
            XMLSchema#integer">
              20000
            </AttributeValue>
          </ResourceMatch>
        </Resource>
      </Resources>
      <Actions> <AnyAction/> </Actions>
    </Target>
    <Condition FunctionId="http://research.sun.com/projects/xacml/names/
    function#time-in-range">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
      time-one-and-only">
        <EnvironmentAttributeDesignator
        DataType="http://www.w3.org/2001/XMLSchema#time"
        AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">
        18:00:00
      </AttributeValue>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">
        23:30:00
      </AttributeValue>
```

```
51      </Condition>
52    </Rule>
53    <Rule RuleId="FinalRule" Effect="Deny"/>
54  </Policy>
```

FIGURA 5.3 – Exemplo de especificação de uma política de acesso.

5.2.4 Geração de *logs*

O módulo de geração de *logs* gera um rastro que apresenta informações sobre as mensagens recebidas e enviadas, permitindo auditar a troca de mensagens e identificando problemas no funcionamento ou na utilização do sistema.

O parâmetro `log_level` define o detalhamento do rastro a ser gerado: baixo, médio ou alto. No nível baixo, apenas o remetente/destinatário, além da data de recebimento/envio das mensagens é registrado. No nível médio, são também registrados o tamanho das mensagens e os módulos de segurança aplicados em cada uma. Por fim, quando configurado com um nível alto de detalhamento, são registrados todos os campos das mensagens trocadas, incluindo os campos de dados da aplicação. Adicionalmente, foi incorporado suporte a um parâmetro `extra_fields`, que permite ao administrador especificar nomes de campos quaisquer a serem registrados, além daqueles já previstos no nível escolhido. Por exemplo, pode-se definir o uso do nível baixo de detalhamento e incluir o registro de um campo hipotético `tipo_de_mensagem`, inserido pela aplicação sendo executada. O rastro gerado é direcionado para um arquivo de saída estabelecido também através de um parâmetro de configuração.

5.3 Assistente para configuração de aspectos de segurança

A criação e a edição do arquivo de configuração definido na Seção 4.7, se feita manualmente, pode representar uma tarefa complexa e que exige um grau de entendimento bastante elevado do formato empregado. Tendo esta dificuldade em mente, foi implementada uma interface gráfica de configuração que permite ao administrador do *peer* preparar este arquivo rapidamente. Adicionalmente, a ferramenta permite interpretar e editar arquivos já existentes, fazendo com que o processo de configuração possa ser gradual.

As Figuras 5.4, 5.5, 5.6 e 5.7 apresentam imagens das diferentes etapas de configuração presentes na ferramenta. A Figura 5.4 apresenta os *peers* identificados na rede

P2P bem como as suas configurações coletadas com o auxílio do módulo de configuração. Além dos *peers* identificados automaticamente na rede, pode-se adicionar manualmente novos *peers*, que eventualmente não estejam presentes na rede. Na Figura 5.5 observa-se a definição dos perfis na qual serão enquadrados os *peers* remotos. Para cada perfil são definidos, inicialmente, os requisitos (`Requirements`) exigidos. A partir desta seleção, a interface apresenta as combinações dos módulos implementados que satisfazem os requisitos escolhidos (`Modules`). Uma vez que na implementação atual não existe multiplicidade de módulos para um mesmo requisito, é apresentada apenas uma combinação de módulos que casa com os requisitos. Além dos perfis definidos pelo administrador, dois outros perfis são automaticamente incluídos: `[broadcast]` (relativo às mensagens em processo de envio por *broadcast*) e `[default]` (utilizado quando o *peer* destino ou origem da mensagem não foi enquadrado em nenhum perfil). A Figura 5.6 ilustra o enquadramento dos *peers* nos perfis existentes. Por fim, na Figura 5.7 são apresentados os módulos disponíveis e os valores de seus parâmetros, já no formato que será utilizado no arquivo de configuração (Seção 4.7).

A configuração dos módulos pode ser realizada de duas formas. A partir da tela que apresenta os módulos disponíveis pode ser realizada uma configuração "global" (Figura 5.8(a)). Os valores definidos desta forma servem como base para todos os perfis definidos. Para o caso de haverem perfis com necessidades diferentes de parâmetros para os módulos empregados, a configuração pode ser feita também individualmente para cada perfil. Neste caso, a configuração é acionada a partir da tela de definição dos perfis (Figura 5.8(b)). Em ambos os casos, a tela para ajuste dos parâmetros é gerada dinamicamente, de acordo as informações contidas no arquivo de caracterização dos módulos (descrito na Seção 4.6).

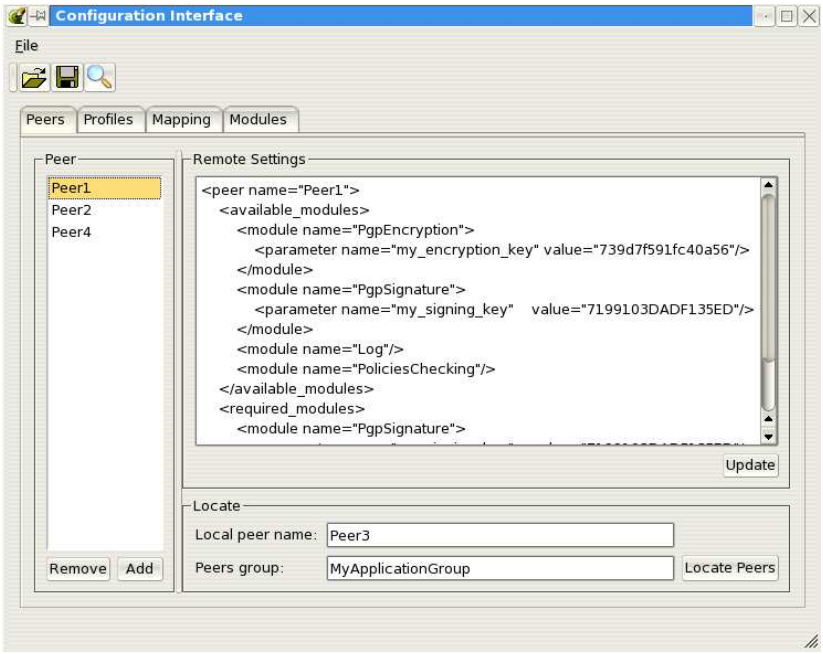


FIGURA 5.4 – Interface de configuração: identificação de *peers* remotos.

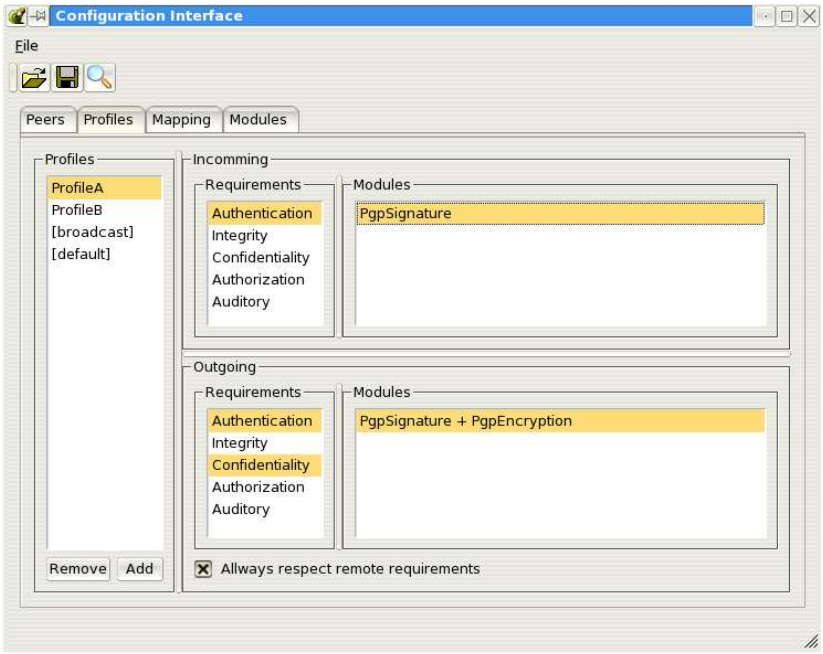


FIGURA 5.5 – Interface de configuração: definição de perfis.

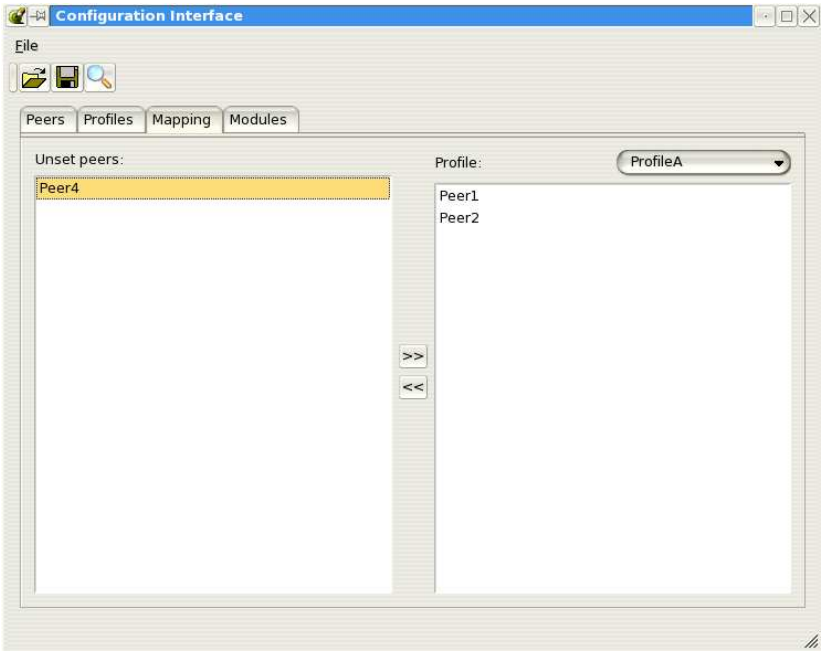


FIGURA 5.6 – Interface de configuração: mapeamento dos *peers* nos perfis definidos.

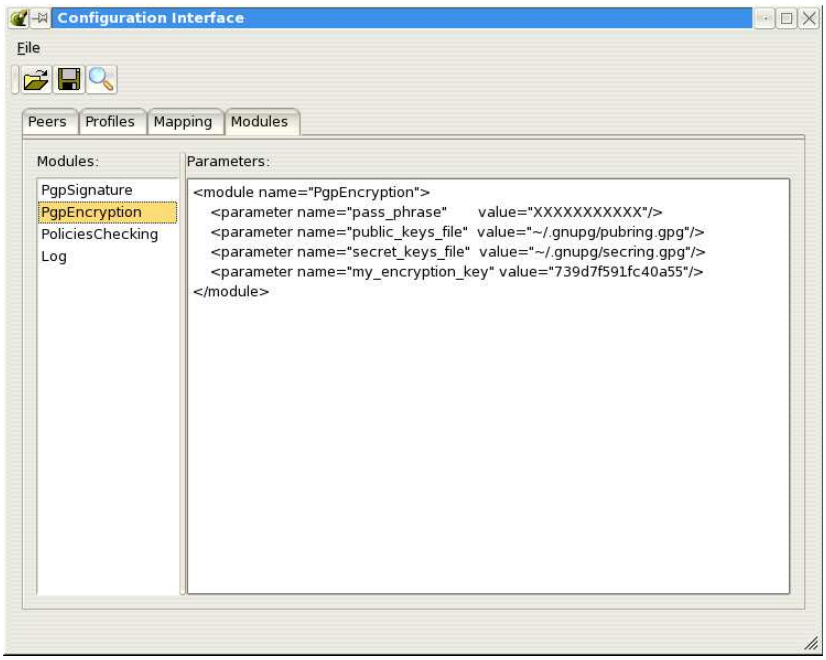
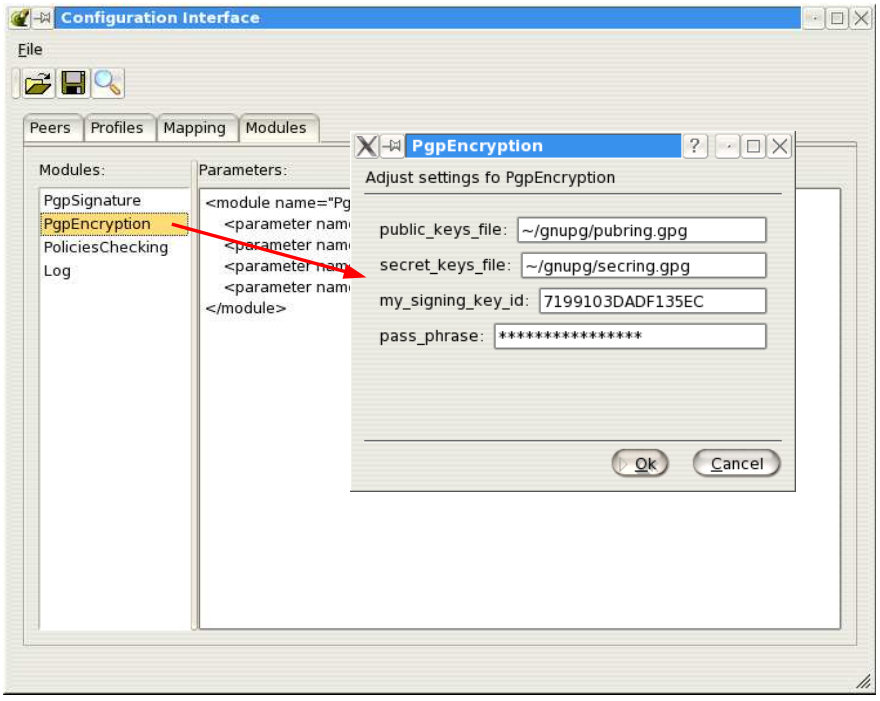
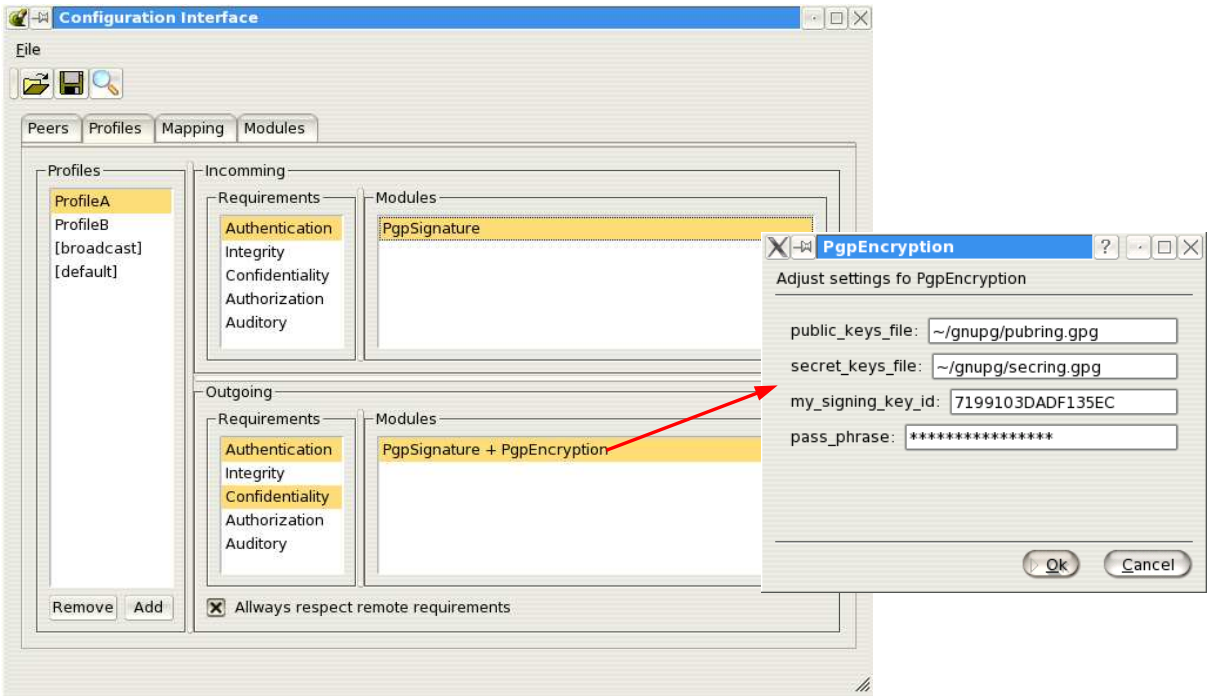


FIGURA 5.7 – Interface de configuração: configuração dos módulos disponíveis.



(a) Configuração global de um módulo.



(b) Configuração de um módulo para um perfil específico.

FIGURA 5.8 – Opções para configuração de um módulo de segurança.

Capítulo 6

Avaliação Experimental

A avaliação experimental da arquitetura foi realizada através de sua utilização com o OurGrid, *middleware* P2P para criação de grades computacionais. Este capítulo inicia com uma descrição do funcionamento básico do OurGrid (Seção 6.1). Em seguida, na Seção 6.2, é descrita a incorporação da arquitetura proposta ao OurGrid e apresentada uma avaliação do impacto resultante. Na Seção 6.3, são apresentados experimentos de avaliação de sobrecarga onde cada módulo implementado é avaliado com o intuito de observar o tempo de processamento necessário para sua utilização. Por fim, a Seção 6.4 apresenta uma breve análise adicional dos resultados obtidos, considerando os objetivos estabalecidos no início da dissertação e os tipos de aplicações P2P citado na Seção 2.1.

6.1 OurGrid

O OurGrid é um *middleware* baseado em P2P que permite a criação de grades computacionais inter-institucionais para a execução de aplicações *bag-of-tasks* ([73]). Conforme ilustrado na Figura 6.1, o OurGrid segue um modelo hierárquico, onde grades locais (por exemplo, A e B), cada uma composta de máquinas pertencentes a uma única organização, são ligadas através de uma rede P2P. Cada *peer* da rede é responsável pelo escalonamento de tarefas nas máquinas locais (denominadas *grid machines*) e pela comunicação com os *peers* das outras instituições.

Em um sistema OurGrid, recursos são compartilhados respeitando uma rede de favores descentralizada ([74]): uma organização provê recursos para outra com a expectativa de ser recompensada no futuro com a execução de suas próprias tarefas remotamente. Quando a demanda por recursos pelos usuários de uma organização (por exemplo, A) ex-

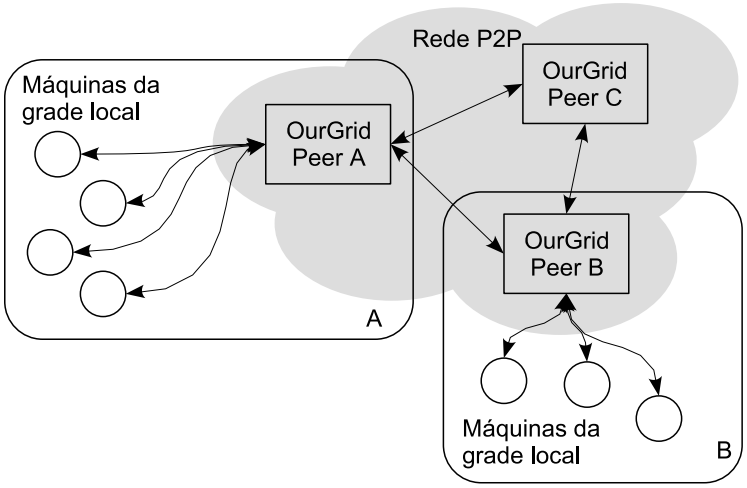


FIGURA 6.1 – Modelo hierárquico de uma grade OurGrid.

cede o poder computacional disponível localmente, seu *peer* propaga (por *broadcast*) uma requisição (mensagem *ConsumerQuery*) aos demais *peers*, conforme ilustrado na etapa 1 da Figura 6.2. Se alguma organização possui recursos ociosos que satisfazem os requisitos do conjunto de tarefas a ser executado, a requisição é respondida através de uma mensagem *ProviderWorkRequest* (2). No caso da Figura 6.2, a organização B responde à requisição, enquanto a organização C não transmite nenhuma mensagem de resposta. A partir do recebimento de uma ou mais respostas, o Peer A escolhe a organização que vai realizar o processamento das tarefas. Inicia-se então uma troca de mensagens com o *peer* escolhido, o Peer B no exemplo, referente à preparação para a execução da tarefa. Essa fase envolve a criação de um espaço temporário na *grid machine* onde a tarefa será executada e a transferência dos arquivos executáveis e de dados necessários para a execução. A tarefa tem então a sua execução iniciada (5), após a qual se dá o início da fase de obtenção dos dados resultantes (7). Por fim, o Peer A envia uma nova mensagem por *broadcast*, informando que *grid machines* ociosas não são mais necessárias para atender a requisição realizada inicialmente, finalizando o processo (8). Apesar de existirem sistemas de troca de recursos mais sofisticados, como o *economic model* descrito em [75], o esquema adotado pelo OurGrid é suficiente para desencorajar *free-riding* (uso de recursos sem retribuição aos outros *peers*) e para prover compartilhamento justo de recursos ([76]).

A implementação do OurGrid já se encontra em um estágio avançado de desenvolvimento. Entretanto, a incorporação de mecanismos de segurança ainda é uma questão

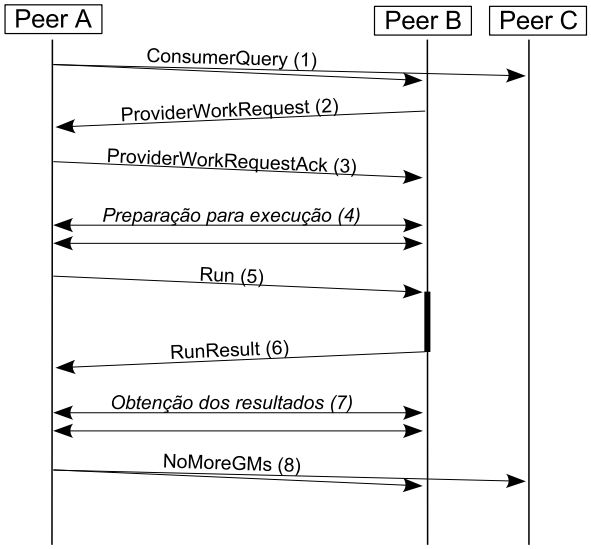


FIGURA 6.2 – Protocolo utilizado pelo OurGrid para execução remota de tarefas.

em aberto ([77]). Isto faz do OurGrid um alvo bastante adequado para o uso da arquitetura aqui proposta, sobretudo considerando a demanda de requisitos de segurança existentes em aplicações de grades computacionais (Seção 2.3). A seção a seguir discute esta incorporação.

6.2 Incorporação da arquitetura de segurança ao OurGrid

A implementação do OurGrid[1] concentra as funções de comunicação entre os *peers*, em especial, `sendMessage`, `broadcast` e `receiveMessage`, em um classe específica, chamada `JALCommunicator`. Essa classe, por sua vez, acessa os métodos da classe `EZMinimalPeer`, provida pelo JAL (Seção 3.3). A adição da camada de segurança ao código do OurGrid (Figura 6.3) se dá pela simples substituição do uso da classe `EZMinimalPeer` pelo uso da classe `SecurePeer` implementada neste projeto. Do ponto de vista de programação, esta incorporação se resume à alteração da linha de código relativa à instanciação do objeto usado na comunicação, o que demonstra o isolamento existente entre a arquitetura proposta e a aplicação P2P.

Uma vez aplicada a camada de segurança ao OurGrid, foi criado um cenário prático de uso em um ambiente controlado, com o objetivo de observar o funcionamento da

[1]Foi utilizada como base o OurGrid versão 0.7 alpha.

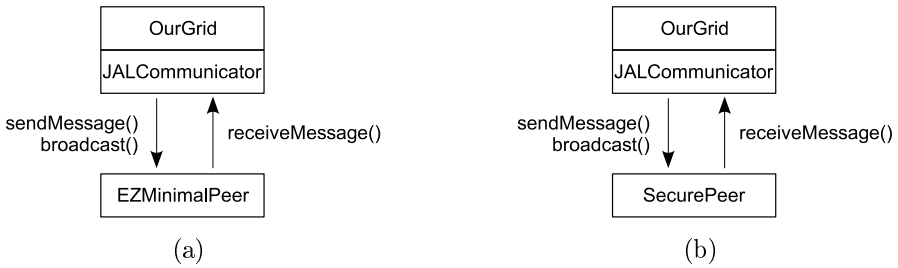


FIGURA 6.3 – Camada de comunicação do OurGrid antes (a) e depois (b) da utilização da camada de segurança.

arquitetura e avaliar o impacto gerado sobre o protocolo de troca de tarefas do OurGrid. O cenário idealizado é composto de quatro *peers*, numerados de 1 a 4. A Figura 6.4 ilustra de forma geral este cenário. O Peer 1 representa uma instituição com tarefas computacionais pendentes, e sem recursos disponíveis. Em função disso, foi instanciado um *peer* que não controla nenhuma *grid machine*, mas a partir do qual partem as requisições de execução. Os *peers* 1, 2 e 4 implementam a arquitetura de segurança, mas enquanto as configurações dos *peers* 1 e 2 permitem a comunicação mútua, o Peer 4 foi configurado de forma a barrar as mensagens provenientes do Peer 1, através da definição de uma política de acesso com este fim. Por fim, o Peer 3 não implementa a camada de segurança, representando uma instância inalterada do OurGrid.

O cenário acima descrito permite ilustrar e analisar:

- a possibilidade de implantação gradual da arquitetura de segurança através da participação de *peers* que não implementam a arquitetura, como no caso do Peer 3;
- o descarte de mensagens por parte da arquitetura em função de falha no atendimento de um requisito de segurança, como no caso do Peer 4, onde o módulo de verificação de políticas é configurado de forma a não autorizar o recebimento de mensagens do Peer 1;
- a heterogeneidade de configurações de segurança em um *peer*, característica observada no Peer 1, que foi configurado com dois perfis: o primeiro relativo a *peers* que implementam a camada de segurança (*peers* 2 e 4) e o segundo relativo a *peers* que não a implementam (Peer 3);
- a sobrecarga gerada pela camada de segurança em cada troca de mensagens entre os *peers*.

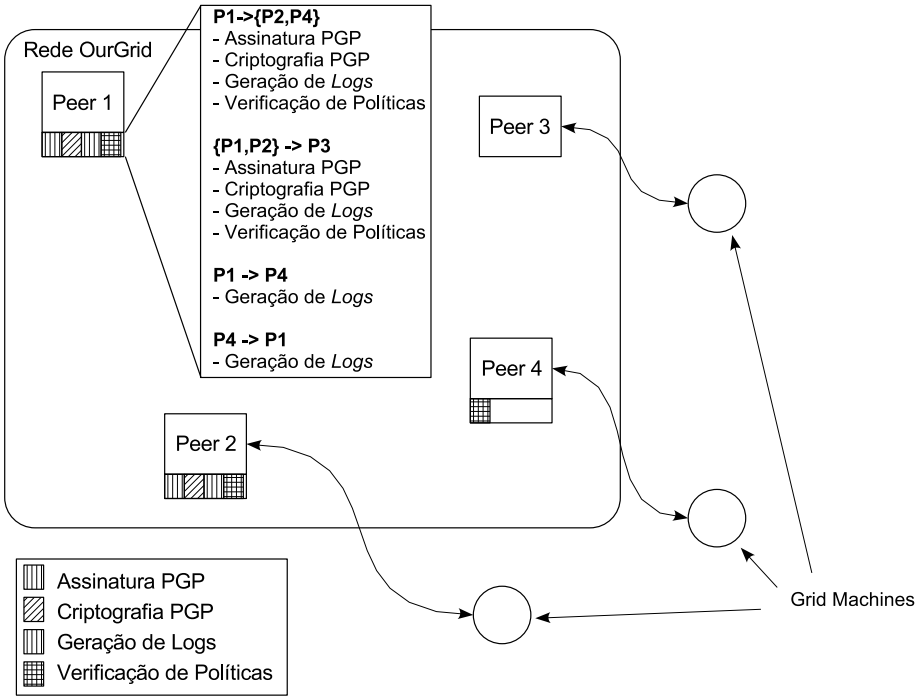


FIGURA 6.4 – Cenário utilizado no experimento.

As tarefas computacionais utilizadas correspondem à simulação de um modelo de cinética intracelular viral baseada em [78]. Mais informações sobre as tarefas e a implementação do modelo aqui utilizado podem ser encontradas em [79]. O experimento foi executado em máquinas Pentium 4 de 2.4 GHz, com 1GB de memória RAM, conectados através de uma rede *Fast Ethernet*. A arquitetura de segurança dos *peers* 1 e 2 foi configurada de forma que fossem empregados na comunicação entre ambos *peers* todos os quatro módulos implementados e descritos na Seção 5.2. Foram utilizadas configurações típicas para cada um destes módulos: módulo de assinatura PGP utilizando chave *DSA* de 1024 bits, criptografia PGP com chave *ElGamal* de 1024 bits, módulo de verificação de políticas abrangendo apenas um papel e uma política (referente ao período do dia em que mensagens podem ser recebidas) e geração de *logs* em nível médio (apresentando apenas o remetente ou destinatário, módulos aplicados, tamanho da mensagem e instante de envio / recebimento). Já para o caso de mensagens trocadas com o Peer 3 (que não implementa a camada de segurança), o Peer 1 foi configurado para empregar apenas o módulo de geração de *logs*, utilizando porém o nível máximo de detalhamento.

As Figuras 6.5 e 6.6 apresentam o diagrama referente às trocas de mensagens e os

tempos envolvidos em cada operação. Para todos os *peers*, foi aferido o tempo total entre o recebimento de uma mensagem e o envio da próxima (resultante do processamento daquela que foi recebida). Foi medido também o tempo despendido pela camada de segurança para a aplicação de todos os módulos no envio e no recebimento das mensagens (tempo entre parênteses). Pode-se observar todas as fases do processo de troca de favores existente no OurGrid, indicadas por setas pontilhadas laterais aos *peers* 2 e 3. Para simplificar a representação, facilitando o entendimento do processo, o diagrama não apresenta a troca de mensagens de `Ping` e `PingResult`, utilizadas para verificar se as *grid machines* remotas continuam acessíveis. Por não ser afetado pela camada de segurança e não representar fator relevante para este trabalho, o tempo de comunicação também foi abstraído.

Na fase de negociação, o Peer 1 envia uma mensagem de `ConsumerQuery` requisitando recursos para execução das tarefas pendentes. A mensagem chega aos três outros *peers*, mas é descartada pela camada de segurança do Peer 4, que não a entrega localmente ao OurGrid. Esta mensagem é respondida pelos *peers* 2 e 3 com uma mensagem `ProviderWorkRequest`, informando que há recursos para a execução das tarefas nesses *peers*. O Peer 1 passa então a enviar mensagens de `Run` e `Put`[2] para os *peers* 2 e 3, preparando o ambiente de execução nas *grid machines* controladas por esses *peers*. Nesta fase, é criado o diretório temporário onde a tarefa será executada e são transferidos os arquivos necessários para a execução da mesma.

Terminado o processo de preparação dos ambientes de execução nas *grid machines*, tem inicio a execução das tarefas, ordenada por uma mensagem de `Run`. O tempo de execução das tarefas é de aproximadamente 180 e 220 segundos (nos *peers* 2 e 3, respectivamente). Essa diferença entre os tempos de execução faz com que, logo após a primeira tarefa tenha sido concluída (na *grid machine* controlada pelo Peer 2) e seus dados coletados, uma réplica da tarefa ainda em execução na *grid machine* controlada pelo Peer 3 seja repassada ao Peer 2, seguindo a política de escalonamento redundante utilizada no OurGrid. Para evitar o excesso de informações no diagrama da Figura 6.6, as mensagens individuais referentes a este processo de re-escalonamento não foram representadas.

Após o término da execução da tarefa na *grid machine* controlada pelo Peer 3, os dados resultantes são transferidos para o Peer 1 e a execução da réplica desta tarefa no Peer 2 é cancelada, através de uma mensagem de `Kill`. Realizada esta operação,

[2]Mensagens de `Put`, `PutResult`, `Run`, `RunResult`, `Kill`, `KillResult`, `Get` e `GetResult` são referenciadas pelo OurGrid como sendo do tipo `GridMachineUse`, por representarem mensagens destinadas à *grid machine* alvo da tarefa. Para facilitar o entendimento do processo, entretanto, os diagramas aqui apresentados e as explicações correspondentes referenciam tais mensagens através do seu sub-tipo interno.

uma mensagem de `NoMoreGMs` é enviada a todos os *peers* (por *broadcast*), informando que a requisição referente à mensagem de `ConsumerQuery` original já foi completamente atendida.

Observa-se que a sobrecarga gerada pela camada de segurança representa, na maioria das operações de processamento das mensagens entre os Peers 1 e 2, entre 30% e 60% da tempo de processamento total. No caso das mensagens enviadas do Peer 1 para o Peer 3, onde apenas a geração de *log* estava ativada, a sobrecarga gerada pela arquitetura de segurança representa entre 5% e 10% do processamento das mensagens. Estima-se que o atraso gerado não afete significativamente o desempenho global do sistema, considerando a granulosidade do paralelismo nas aplicações de grade (ou seja, a predominância dos tempos de processamento das tarefas em relação às trocas de mensagens). Especificamente no caso da avaliação realizada, a execução remota das tarefas computacionais levou em torno de 182 segundos para uma tarefa e 220 segundos para a outra.

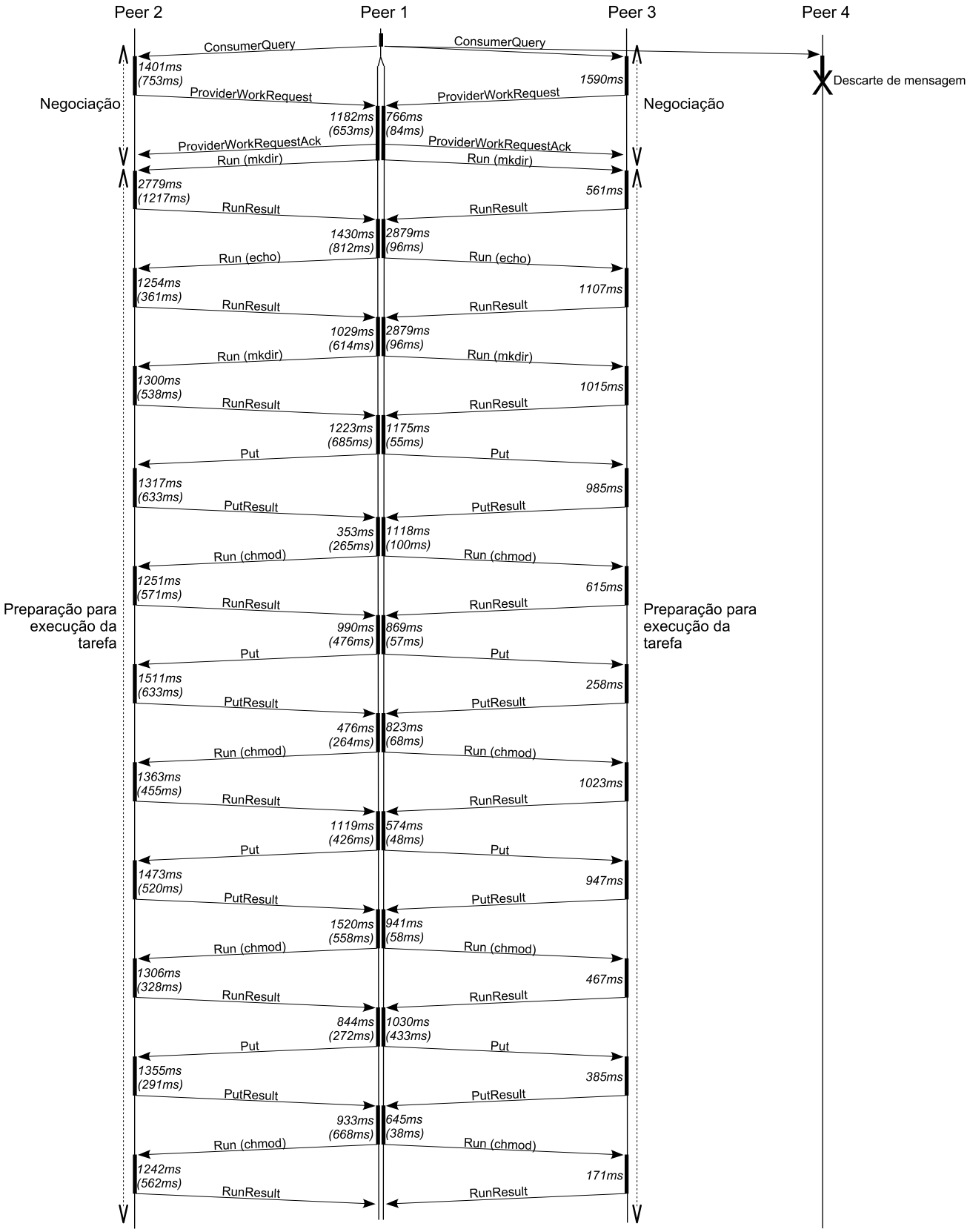


FIGURA 6.5 – Troca de mensagens entre *peers* Ourgrid.

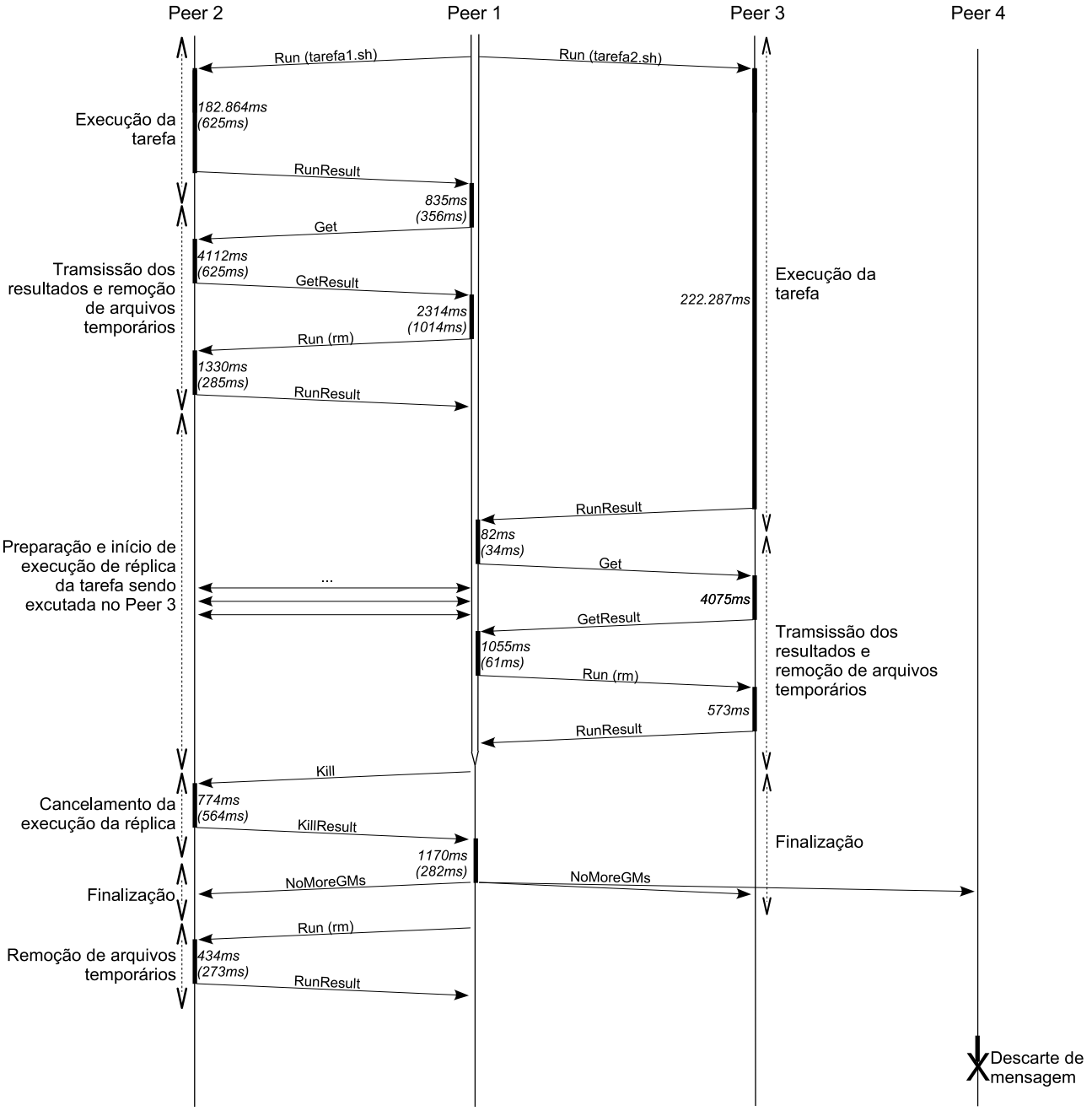


FIGURA 6.6 – Troca de mensagens entre *peers* Ourgrid *(cont.)*.

6.3 Avaliação de sobrecarga

Apesar de a sobrecarga da arquitetura em aplicações de grades computacionais ser proporcionalmente baixa frente ao consumo total de recursos, esse pode não ser o caso em outras aplicações. Com o objetivo de analisar a sobrecarga decorrente da utilização da arquitetura proposta em aplicações P2P de forma geral, o protótipo desenvolvido foi avaliado através de experimentos específicos. Foram avaliadas a sobrecarga do uso da camada de segurança e de cada módulo individualmente através de uma aplicação sintética. O cenário de experimentação utilizado continha apenas dois *peers*: um transmissor e um receptor. Os dados foram obtidos através do envio e recebimento de mensagens de 1, 10 e 50KB, contendo campos de texto gerados aleatoriamente com tamanho fixo em 1KB. Os resultados apresentados a seguir representam os tempos médios obtidos após a execução de 50 transmissões e recebimentos de mensagens.

Camada de segurança

A Figura 6.7 apresenta os resultados referentes a sobrecarga gerada pela camada de segurança sem a utilização de nenhum módulo. Observa-se uma sobrecarga de aproximadamente 30 ms em cada operação, sem grande variação com o aumento do tamanho da mensagem.

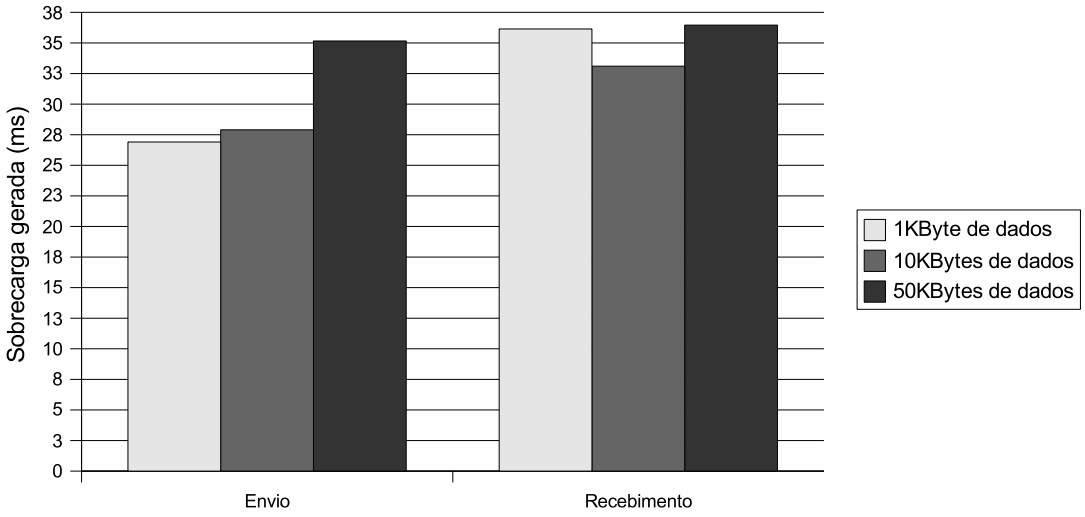


FIGURA 6.7 – Sobrecarga gerada no envio e recebimento de mensagens pela camada de segurança sem o uso de nenhum módulo.

Módulo de assinatura PGP

Para avaliação do módulo de assinatura PGP, foram empregadas chaves DSA (*Digital Signature Algorithm*) e RSA (*Rivest Shamir Adleman*), com diferentes tamanhos. No de DSA, foram utilizadas chaves com tamanhos de 768 e 1024 bits, respectivamente os tamanhos mínimo e máximo permitido por esta técnica. Para RSA, foram empregados o valor mínimo, 1024 bits, o valor máximo recomendado, 2048 bits, e também 4096 bits, um valor acima do máximo recomendado (devido à sobrecarga de processamento gerada), mas ainda assim utilizado por alguns usuários.

Ao contrário do que acontece no uso da camada de segurança sem nenhum módulo, a sobrecarga gerada pela utilização do módulo de assinatura PGP apresenta uma elevação significativa com o aumento do tamanho das mensagens processadas. Conforme ilustrado nas Figuras 6.8 e 6.9, o tempo de processamento para geração e verificação de assinaturas DSA de mensagens de 50 KB fica entre 80 e 100 ms seja com tamanho de chave de 768 ou 1024 bits. No caso de chaves RSA, existe uma grande variação na sobrecarga de processamento relacionada à geração das assinaturas em função do tamanho da chave. Enquanto para um tamanho de 1024 bits o tempo de processamento se assemelha ao caso de uso de chave DSA, este tempo chega a quase 200 ms para uma chave de 2048 bits (considerando mensagens com 50KB de dados) e ultrapassa 800 ms quando uma chave de 4096 bits é utilizada. Em contrapartida, essa grande variação no tempo de processamento em função do tamanho de chave não acontece na fase de recebimento de mensagens, onde a assinatura é verificada. O tempo para verificação de assinaturas RSA de mensagens com 50 KB de dados fica em torno de 60 ms, mesmo com o uso de chaves de 4096 bits.

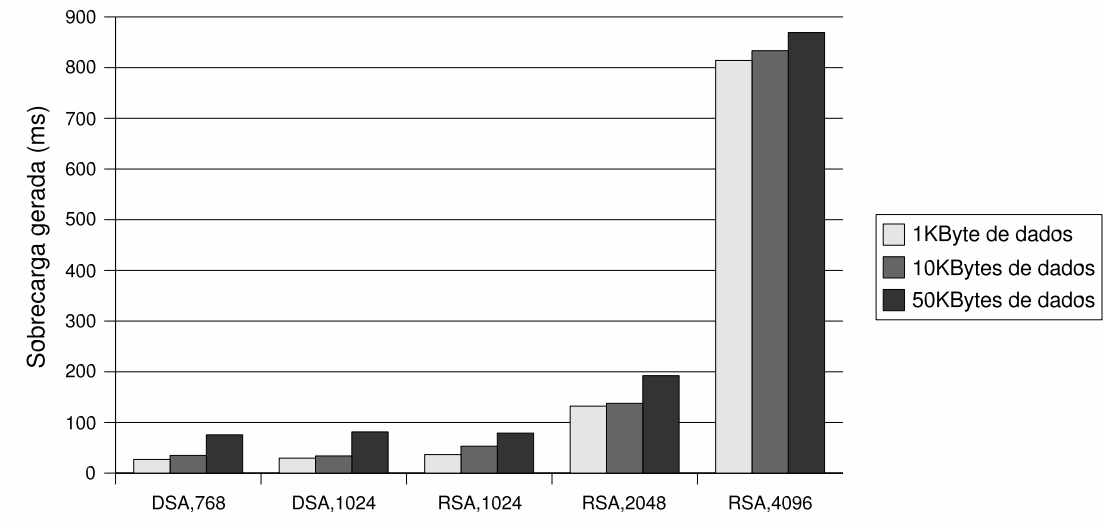


FIGURA 6.8 – Sobrecarga gerada no cômputo de assinaturas.

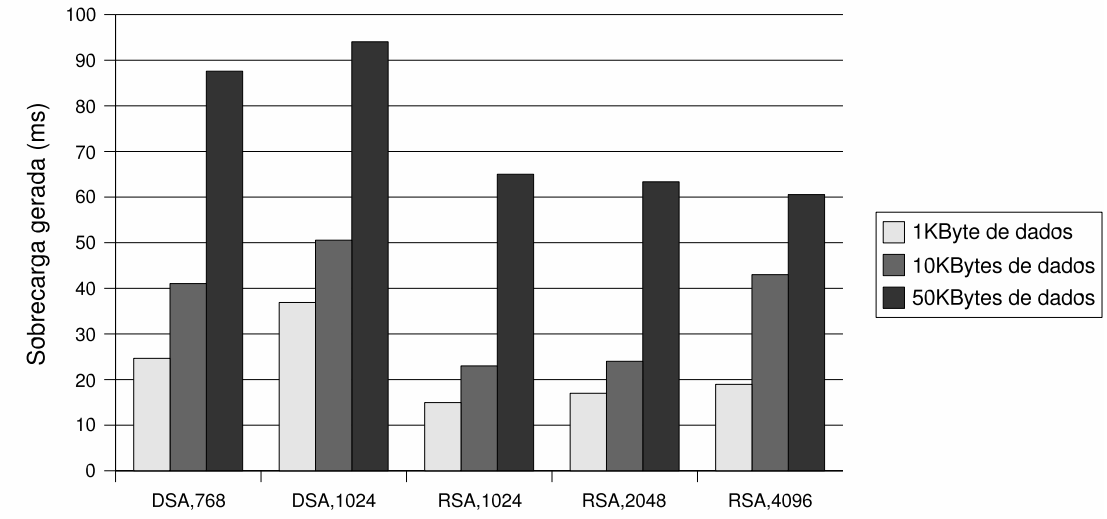


FIGURA 6.9 – Sobrecarga gerada na verificação de assinaturas.

Módulo PgpEncryption

O módulo de cifragem PGP foi avaliado através do uso de chaves *ElGamal* com tamanhos de 768, 1024 e 2048 bits, valores mínimo, típico e máximo, respectivamente. Através das Figuras 6.10 e 6.11 observa-se que, dentre os módulos implementados, e con-

siderando configurações típicas de uso, a cifragem PGP é a etapa que representa a maior sobrecarga. Considerando uma chave com tamanho de 1024 bits, mesmo mensagens com 1 KB de dados apresentam uma sobrecarga de aproximadamente 120 ms. Na etapa de decifragem, realizada no recebimento das mensagens, esta sobrecarga é menor. Mensagens com 1KB de dados levam, em média, 75 ms para serem processadas nesta etapa (considerando uma chave *ElGamal* de 1024 bits).

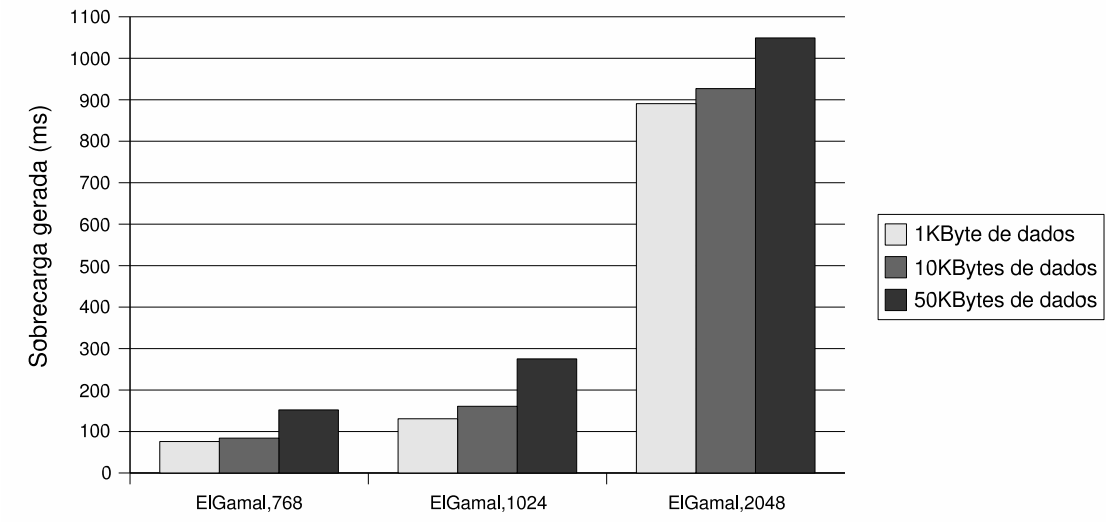


FIGURA 6.10 – Sobrecarga gerada na cifragem de mensagens.

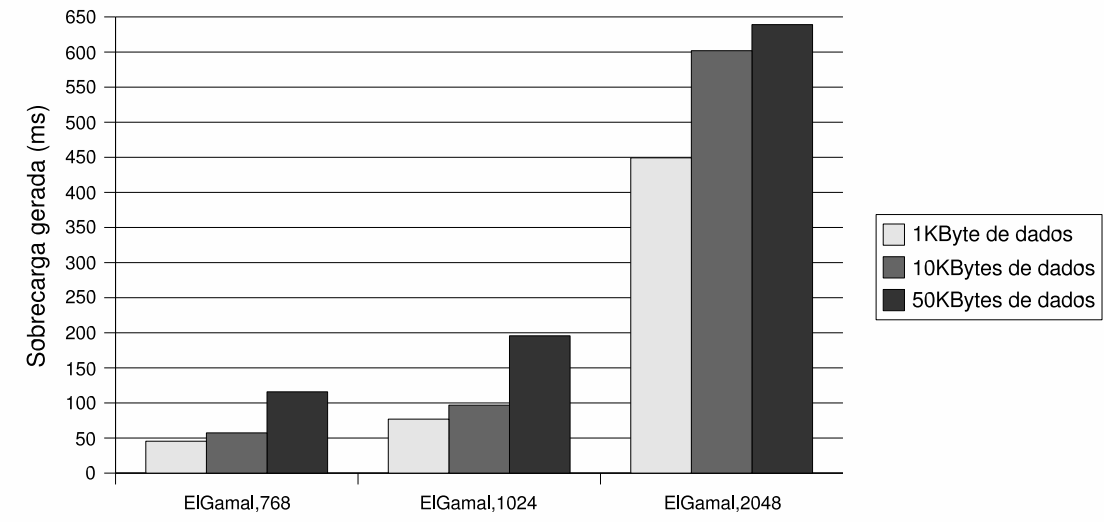


FIGURA 6.11 – Sobrecarga gerada na decifragem de mensagens.

Módulo de geração de *logs*

As Figuras 6.12 e 6.13 apresentam a sobrecarga resultante da geração de *logs* utilizando os níveis de detalhamento baixo (*low*), médio (*medium*) e alto (*high*). Seja no envio ou no recebimento de mensagens, a sobrecarga gerada a partir da configuração de detalhamento baixo e médio fica em torno de 10 ms, para qualquer um dos tamanhos de mensagem avaliados. Já no caso do uso de um nível alto de detalhamento, onde todo o conteúdo das mensagens é armazenado no arquivo de *log*, existe um aumento considerável na sobrecarga de processamento de acordo com o tamanho da mensagem, chegando a 60 ms no caso do recebimento de mensagens de 50 KB.

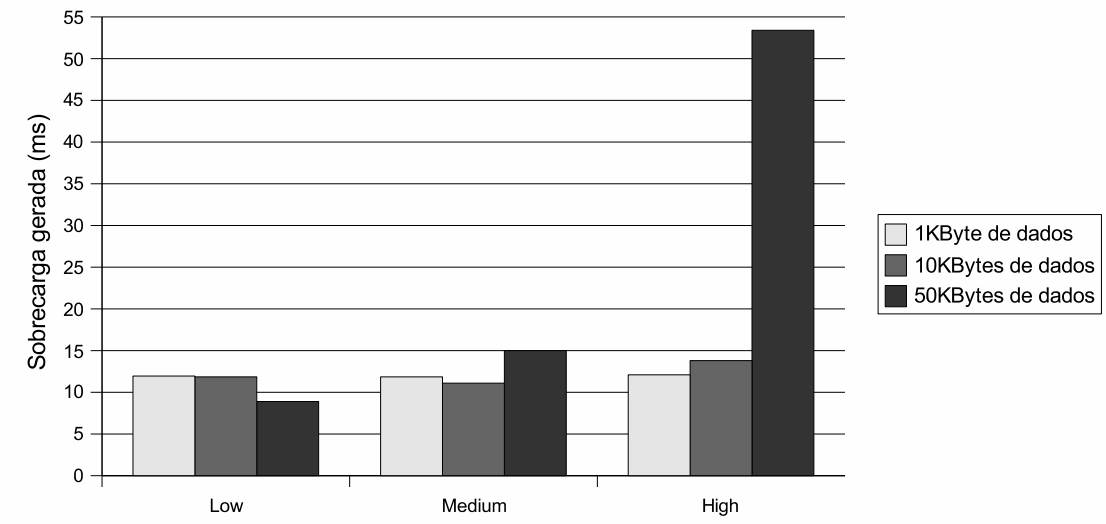


FIGURA 6.12 – Sobrecarga resultante na geração de *log* no envio de mensagens.

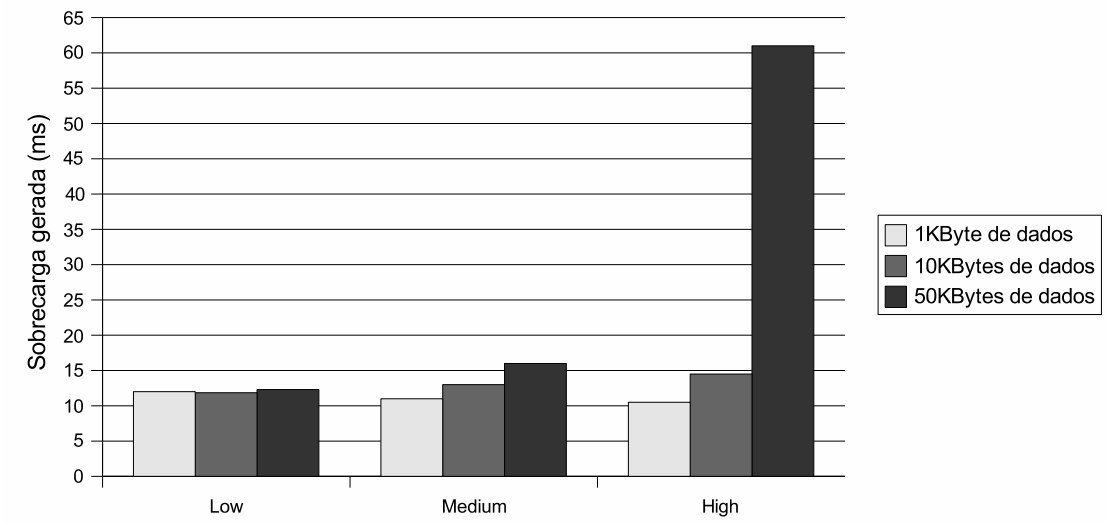


FIGURA 6.13 – Sobrecarga resultante na geração de *log* no recebimento de mensagens.

Módulo de verificação de políticas

O módulo de verificação de políticas apresenta uma sobrecarga de processamento proporcional ao número de políticas presentes no arquivo que define o controle de acesso. A Figura 6.14 ilustra este comportamento. Enquanto no caso de haver apenas uma política

de acesso estabelecida a sobrecarga fica em torno de 3 ms, este valor sobe para até 43 ms no caso de 100 políticas. Observa-se ainda que o tamanho da mensagem não influencia muito na sobrecarga gerada por este módulo.

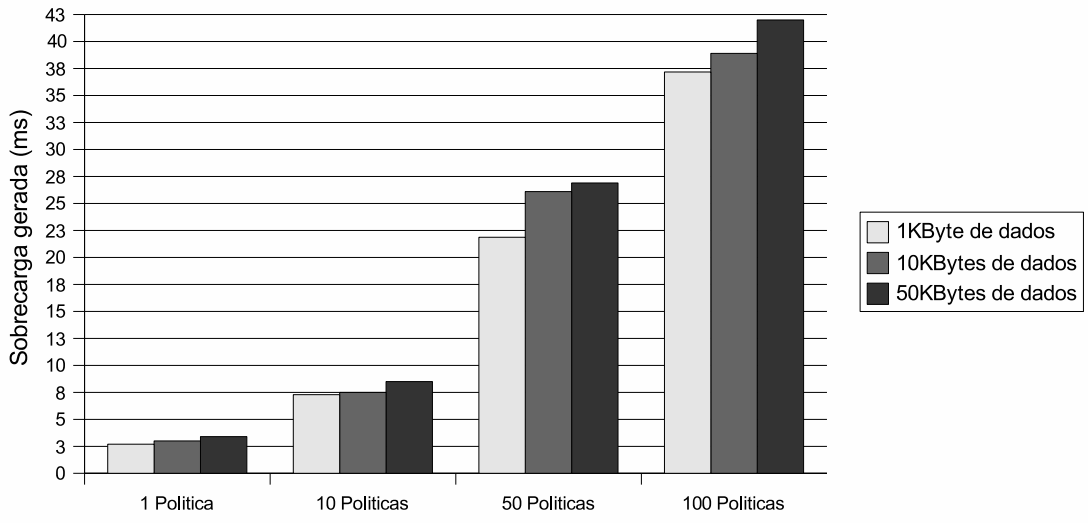


FIGURA 6.14 – Sobrecarga gerada na verificação de políticas.

6.4 Análise dos resultados

Através dos experimentos realizados neste capítulo, pôde-se observar o impacto e a sobrecarga resultante do uso arquitetura em ambientes reais. Inicialmente, comprovou-se a adequação da arquitetura quanto aos objetivos estabelecidos na Seção 1.3, referentes ao isolamento da mesma em relação à aplicação, à modularidade dos mecanismos de segurança e à possibilidade de implantação gradual da solução (considerando um ambiente onde nem todos os *peers* a implementam). Em relação à sobrecarga gerada pela arquitetura, ficou clara a relação entre o grau de segurança desejado e o nível de processamento adicionado pelo uso dos mecanismos. Além de uma seleção cuidadosa de quais técnicas serão empregadas, a configuração dessas técnicas (representada pelos parâmetros dos módulos descritos) também representa um fator bastante relevante no processo de implantação de uma aplicação segura.

Enquanto em aplicações como as de grades computacionais a sobrecarga gerada pela arquitetura tende a não representar um empecilho se considerarmos o funcionamento do

sistema como um todo, em outros tipos de aplicações este processamento adicional pode ser bastante relevante. Analisando os tipos de aplicações P2P enumerados na Seção 2.1, identifica-se duas características que podem dificultar na prática o uso de algoritmos de segurança que tenham uma demanda relativamente grande em termos de processamento: necessidade de interatividade e volume de mensagens característico das aplicações. A primeira característica (interatividade) se faz presente principalmente em aplicações de trabalho cooperativo que apresentem funcionalidades como de vídeoconferência. Nesses casos, a adição de uma sobrecarga da ordem de centenas de milisegundos representa uma desvantagem bastante significativa. Essa característica não inviabiliza o uso de uma arquitetura de segurança como a definida nesta dissertação, mas exige um ajuste mais refinado das técnicas utilizadas, seja pela escolha de algoritmos menos custosos, seja pela utilização de parâmetros que levem a um menor processamento (como por exemplo, chaves de cifragem menores). Aplicações de jogos on-line, dependendo de suas características específicas, também podem apresentar esta limitação quanto ao uso de algoritmos de segurança. O outro aspecto relevante, referente ao volume de mensagens gerado pela aplicação, fica aparente especialmente em aplicações de compartilhamento de arquivos. Apesar de a transferência dos arquivos em si não ser muito afetada pelo atraso adicional gerado pela arquitetura de segurança, essas aplicações tipicamente apresentam um volume de mensagens bastante elevado no que diz respeito aos protocolos de indexação e localização de dados. Considerando esta característica, a utilização de técnicas que acarretem um volume de processamento considerável a cada mensagem pode facilmente sobrecarregar as máquinas onde a aplicação está sendo executada. Aqui, novamente, faz-se necessária uma decisão cuidadosa quanto aos mecanismos empregados. Por fim, aplicações de troca de mensagens não apresentam nem um volume grande de mensagens nem possuem restrições quanto à adição de um atraso adicional de, por exemplo, 300 ms a cada mensagem.

Capítulo 7

Considerações Finais

A segurança em aplicações P2P é um dos fatores indispensáveis para que a sua adoção possa se expandir em ambientes como o corporativo. Esta dissertação apresentou uma arquitetura de segurança modular, extensível e independente de aplicação que visa a atender essa demanda. São disponibilizados diversos aspectos de segurança através de módulos independentes, que podem ser estendidos pelo usuário. A extensibilidade do projeto permite a adição de novos módulos, que podem ser genéricos, fornecendo potencialmente novos aspectos (como não-repúdio ou reputação), ou específicos para alguma determinada aplicação. A arquitetura desenvolvida permite que o processo de adição de novos módulos não dependa de sua recompilação.

Cada *peer* é configurado para empregar um subconjunto dos módulos de segurança implementados, que pode ser especificado de forma independente em cada *peer*. Esta abordagem flexível permite ainda que a configuração de segurança de um *peer* reflita graus variados de confiança em canais de comunicação e *peers* remotos. Para simplificar o processo de configuração, é possível especificar perfis de segurança e atribuí-los a diferentes grupos de *peers*. O uso da arquitetura proposta é transparente, evitando que o desenvolvedor da aplicação tenha que lidar com a programação e o tratamento dos aspectos de segurança.

A implementação da arquitetura foi realizada tendo JXTA/JAL como base, o que permite que ela seja usada em uma grande variedade de projetos já existentes. A arquitetura foi incorporada experimentalmente ao OurGrid, aplicação para criação de grades computacionais baseada no paradigma P2P ([80]). Foi observada a sobrecarga gerada no protocolo de troca de tarefas, indicando que a nossa proposta pode ser utilizada sem representar um acréscimo significativo no tempo total de execução das tarefas na grade.

Adicionalmente, avaliou-se o uso dos módulos de segurança isoladamente, resultados que servem de base para determinação do equilíbrio entre desempenho e nível de segurança desejado, para aplicações P2P quaisquer.

Como trabalhos futuros, planeja-se a avaliação da arquitetura em outros tipos de aplicações P2P, como compartilhamento de arquivos e trabalho colaborativo. Serão também desenvolvidos novos módulos de segurança, ampliando as funcionalidades oferecidas. Para o caso do OurGrid, planeja-se a incorporação de um módulo de verificação de políticas específico que considera a semântica das mensagens trocadas na decisão. A implementação será disponibilizada como software livre, licenciado sob a GPL.

Bibliografia

[1] Z. Ge, D. Figueiredo, S. Jaiswal, J. F. Kurose, and D. Towsley. Modeling peer-peer file sharing systems. In *INFOCOM 2003*, March 2003.

[2] C. Lowth. Securing your network against Kazaa. *Linux Journal*, (114), October 2003.

[3] W. Kim, S. Graupner, and A. Sahai. A Secure Platform for Peer-to-Peer Computing in the Internet. In *35th Hawaii International Conference on System Sciences (HICSS-35 2002)*, January 2002.

[4] The Peer-to-Peer Trusted Library. http://sourceforge.net/projects/ptptl.

[5] G. Lawton. Is Peer-to-Peer Secure Enough for Corporate Use? *IEEE Computer*, 37(1):22–25, January 2004.

[6] D. F. Ferraiolo and D. R. Kuhn. Role-Based Access Controls. In *15th NIST-NSA National Computer Security Conference*, September 1992.

[7] S. Capkun, L. Buttydn, and J. P. Hubaux. Small Worlds in Security Systems: an Analysis of the PGP Certificate Graph. In *New Security Paradigms Workshop*, September 2002.

[8] L. Gong. JXTA: A network programming environment. *IEEE Internet Computing*, 5:88–95, June 2001.

[9] Ourgrid. http://www.ourgrid.org.

[10] J. Rocha, M. Domingues, A. Callado, E. Souto, G. Silvestre, C. Kamienski, and D. Sadok. Peer-to-Peer: Computação Colaborativa na Internet, Minicurso. In *XII Simpósio Brasileiro de Redes de Computadores*, May 2004.

[11] IRTF Peer-to-Peer Research Group. http://www.irtf.org/charters/p2prg.html.

[12] E. Cohen and S. Shenker. Replication Strategies in Unstructured Peer-to-Peer Networks. In *SIGCOM 2002*, pages 177–190, August 2002.

[13] Q. Lv and et al. Search and Replication in Unstructured Peer-to-Peer Networks. In *16o ACM International Conference on Supercomputing (ICS'02)*, June 2002.

[14] Kazaa Project. http://www.kazaa.com.

[15] Q. Lv, S. Ratnasamy, and S. Shenker. Can Heterogeneity make Gnutella Scalable? In *IPTPS 2002*, March 2002.

[16] J. Xu. On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks. In *INFOCOM 2003*, March 2003.

[17] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *SIGCOM 2001*, pages 149–160, 2001.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOM 2001*, pages 161–172, August 2001.

[19] JXTA-whiteboard project. http://whiteboard.jxta.org.

[20] Icq. http://www.icq.com.

[21] Jabber Project. http://www.jabber.org.

[22] M. Waldman, A. D. Rubin, and L. F. Cranor. Publius: A Robust, Tamper-evident, Censorship-resistant,Web Publishing System. *Nineth Security Symposium, USENIX*, pages 59–72, August 2000.

[23] I. Clarke and S. Miller. Protecting Freedom of Information Online with Freenet. *IEEE Internet Computing*, 6:40–49, February 2002.

[24] Groove. http://www.groove.net.

[25] Consilient. http://www.consilient.com.

[26] Project Skipe. http://ui.skype.com/skype.html.

[27] NetMeeting. http://www.microsoft.com/windows/netmeeting.

[28] Bjorn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-peer support for massively multiplayer games. In *INFOCOM 2004*, volume 1, pages 96–107, March 2004.

[29] N. Daswani, H. Garcia-Molina, and B. Yang. Open problems in data-sharing peer-to-peer systems. In *ICDT 2003*, pages 1–15, January 2003.

[30] W. Stallings. *Network Security Essentials - Applications and Standards*. Prentice-Hall, Inc, 2000.

[31] The DES algorithm illustrated. http://www.aci.net/kalliste/des.htm.

[32] RC4 Encryption Algorithm. http://www.vocal.com/RC4.pdf.

[33] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21:120–126, February 1978.

[34] W. Diffie and M. Hellman. New Directions in Criptography. *IEEE Transactions on Information Theory*, November 1976.

[35] D. Atkin, W. Stallings, and P. Zimmermann. PGP Message Exchange Formats. RFC 1991, August 1999.

[36] R. Rivest. The MD5 Message-Digest Algorithm. RFC, April 1992.

[37] D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, September 2001.

[38] J. S. Park and J. Hwang. Role-based Access Control for Collaborative Enterprise In Peer-to-Peer Computing Environments. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 93–99, 2003.

[39] R. Righi, C. Westphall, and F. R. Pellissari. P2P-Role: Uma Arquitetura de Controle de Acesso Baseada em Papéis para Sistemas Colaborativos Peer-to-Peer. In *IV Workshop em Segurança de Sistemas Computacionais*, pages 285–296, May 2004.

[40] The Anonymizer. http://anonymizer.com.

[41] D. Chaum. Untraceable electronic mail, return address, and digital pseudonyms. *Communications os the ACM 24, 2*, February 1981.

[42] Electronic Frontiers Georgia (EFGA). Anonymous remailer information. http://anon.efga.org/Remailers, February 2002.

[43] P. Syverson, D. M. Goldschlang, and M. G. Reed. Anonymous connections and onion routing. In *IEEE Symposium on Security and Privacy*, pages 44–54, May 1997.

[44] I. Goldberg and A. Shostack. Freedom network 1.0 architecture, November 1999.

[45] M. J. Freedman and R. Morris. Tarzan: a peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002*, November 2002.

[46] E. Damiani, S. D. C. Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A Reputation-Based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. In *CCS 2002*, November 2002.

[47] F. Cornelli, E. Damiani, S. D. C. Vimercati, S. Paraboschi, and Pierangela Samarati. Choosing Reputable Servents in a P2P Network. In *WWW2002*, May 2002.

[48] Poblano A Distributed Trust Model for Peer-to-Peer Networks. http://www.jxta.org/docs/trust.pdf.

[49] OpenSSL Project. http://www.openssl.org.

[50] C. Shirky. Web services and context horizons. *IEEE Computer*, 35(9):98–100, September 2002.

[51] World Wide Web Consortium. http://www.w3.org.

[52] Web Services Architecture. http://www.w3.org/TR/ws-arch.

[53] Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl.

[54] Oasis Universal Description, Discovery and Integration specification, August 2004. http://www.uddi.org/specification.html.

[55] SOAP Specifications, August 2004. http://www.w3.org/TR/soap.

[56] Web Services Security (WS-Security) Specification. http://www-106.ibm.com/developerworks/webservices/library/ws-secure, Junho 2004.

[57] E-Speak. http://www.e-speak.hp.com.

[58] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, Sam Meder, L. Pearlman, and S. Tuecke. Security for Grid Services. In *12th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, June 2003.

[59] Globus Toolkit. http://www.globus.org.

[60] B. C. Neuman and T. Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, 32(9):33–38, September 1994.

[61] S. Tuecke, D. Engert, I. Foster, M. Thompson, L. Pearlman, and C. Kesselman. Internet X.509 Public Key Infrastructure Proxy Certificate Profile. IETF, RFC 3820, June 2004.

[62] S. R. Waterhouse, D. M. Doolin, G. Kan, and Y. Faybishenko. JXTA Search: a distributed search framework for peer-to-peer networks. *IEEE Internet Computing*, 6:68–73, February 2002.

[63] JXTA. http://www.jxta.org.

[64] JXTA Projects in Domain. www.jxta.org/servlets/DomainProjects.

[65] T. Dierks and C. Allen. The TLS Protocol, RFC 2246, January 1999.

[66] JAL - JXTA Abstraction Layer. http://ezel.jxta.org/jal.html.

[67] Y. Kim, D. Mazzocchi, and G. Tsudik. Admission Control in Peer Groups. In *Second IEEE International Symposium on Network Computing and Applications*, page 131, April 2003.

[68] D. Brookshier, D. Govoni, N. Krishnan, and J. C. Soto. *JXTA: Java P2P Programming*. Sams Publishing, March 2002.

[69] The International PGP Home Page. http://www.pgpi.org.

[70] The Bouncy Castle Crypto package. http://www.bouncycastle.org.

[71] S. Godik et al. extensible access control markup language (XACML) version 1.1. committe specification, August 2003.

[72] Sun's XACML Implementation Programmer's Guide, August 2004. http://sunxacml.sourceforge.net/guide.html.

[73] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauvé, F. A. B. da Silva, C. O. Barros, and C. Silveira. Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach. In *ICCP'2003 - International Conference on Parallel Processing*, October 2003.

[74] N. Andrade, W. Cirne, F. V. Brasileiro, and P. Roisenberg. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. In *Job Scheduling Strategies for Parallel Processing, 9th International, Workshop, JSSPP 2003*, pages 61–86, June 2003.

[75] D. Abramson, R. Buyya, and J. Giddy. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems (FGCS)*, 18:1061–1074, August 2002.

[76] N. Andrade, M. Mowbray, W. Cirne, and F. Brasileiro. When Can an Autonomous Reputation Scheme Discourage Free-riding in a Peer-to-Peer System? In *4th International Workshop on Global and Peer-to-Peer Computing*, April 2004.

[77] N. Andrade. Reputação Autônoma como Incentivo à Colaboração no Compartilhamento de Recursos Computacionais. Master's thesis, Universidade Federal de Campina Grande, March 2004.

[78] R. Srivastava and L. You and J. Yin. Stochastic vs. Deteministic Modeling of Intracellular Viral Kinetics. *Theory Biology*, 218:309–321, 2002.

[79] G. B. Bedin and N. Lemke. Arquitetura Computacional para Simulação de Redes metabólicas em Larga Escala. In *I Workcomp*, May 2004.

[80] A. Detsch, L. P. Gaspary, M. Barcellos, and G. G. H. Cavalheiro. Towards a Flexible Security Framework for Peer-to-Peer based Grid. In *2nd International Workshop on Middleware for Grid Computing - MGC 2004 (Co-located with 5th ACM Middleware 2004)*, October 2005.