

**UNIVERSIDADE DO VALE DO RIO DOS SINOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**Escalabilidade, Autonomia e Segurança em Redes
Peer-to-Peer: repensando a P2PSL**

Giovani Facchini

**Prof. Marinho Pilla Barcellos
Orientador**

*Monografia submetida como requisito
parcial para a obtenção do título de
Bacharel em Informática.*

São Leopoldo, Novembro de 2006

Resumo

Peer-to-Peer (P2P) é um conceito em grande expansão cada vez mais utilizado na Internet. Com a grande expansão deste modelo existe a necessidade investigativa dos problemas de segurança inerentes deste modelo e como os mesmos podem ser resolvidos. Esta monografia aborda os conceitos de redes P2P e segurança sobre as mesmas. A camada de segurança P2PSL será explorada e analisada mostrando suas vulnerabilidades e por fim serão feitas propostas na tentativa de diminuir o impacto de ataques contra a segurança das redes P2P.

Sumário

Resumo	2
1 Introdução	5
1.1 Contextualização	5
1.2 Definição dos Problemas	Erro! Indicador não definido.
1.3 Objetivos.....	6
1.4 Organização da Monografia	6
2 Conceitos de Redes Peer-to-Peer	8
2.1 Aplicações Peer-to-Peer	10
2.2 Redes Não-Estruturadas	11
2.3 Redes Estruturadas	14
2.4 Fundamentos de Segurança em Redes Peer-to-Peer.....	16
3 Criptografia e Gerência de Chaves	18
3.1 Criptografia Simétrica	18
3.2 Criptografia Assimétrica.....	20
3.3 Gerenciamento e Troca de Chaves	22
3.3.1 Distribuição de Chaves Secretas	23
3.3.2 Distribuição de Chaves Públicas	25
3.3.3 Distribuição de Chaves Secretas Usando Criptografia Assimétrica.....	32
4 Peer-to-Peer Security Layer.....	35
4.1 Visão Geral.....	35
4.2 Funcionamento Interno.....	36
4.3 Configuração dos Módulos e Requisitos de Segurança.....	37
4.4 Configuração dos Perfis	39
4.5 Troca de Requisitos	41
4.6 Implementação.....	42
5 Análise Crítica da P2PSL.....	44
5.1 Arquitetura.....	44
5.2 Escalabilidade	45
5.3 Segurança.....	46
5.3.1 Negação de Serviço	47
5.3.2 Roteamento.....	47

5.3.3	Confidencialidade.....	48
5.3.4	Autenticidade.....	48
5.3.5	Reputação e Confiança.....	49
5.3.6	Autorização.....	50
5.3.7	Integridade.....	51
5.3.8	Anonimidade e Negabilidade.....	51
5.3.9	Poluição de Conteúdo.....	52
6	Proposta de Melhorias.....	53
6.1	Arquitetura.....	53
6.1.1	P2PSL entre Aplicação e Substrato de Rede.....	53
6.1.2	P2PSL embaixo do Substrato de Rede.....	56
6.1.3	P2PSL Protegendo Aplicação e Substrato de Rede.....	58
6.2	Escalabilidade.....	61
6.2.1	Troca de Chaves.....	61
6.2.2	Troca de Requisitos.....	62
6.3	Segurança.....	62
6.3.1	Negação de Serviço.....	63
6.3.2	Roteamento.....	63
6.3.3	Autenticidade.....	64
6.3.4	Reputação e Confiança.....	64
6.3.5	Autorização.....	64
6.3.6	Anonimidade e Negabilidade.....	65
6.3.7	Poluição de Conteúdo.....	65
7	Implementação – P2PSL 2006.....	67
8	Conclusão.....	68
	Bibliografia.....	69

[GF1] Comentário: Lembrar de retirar o sumário do sumário ;)

1 Introdução

1.1 Contextualização

Nos últimos anos, redes Peer-to-Peer (P2P) aparecem como um paradigma importante de comunicação e de distribuição de dados. Seu crescimento se deve, principalmente, pela popularização de aplicativos para compartilhamento de arquivos.

O aplicativo que iniciou esse processo foi o Napster [39], que permitia que usuários trocassem arquivos de música em formato MP3 alheios a questões de *copyright*. Dado o enorme crescimento visto em número de usuários, logo surgiram outras alternativas para compartilhamento de arquivos. Apesar de Napster não adotar uma filosofia P2P na sua essência, o crescimento desta aplicação reacendeu a questão da autonomia e interação direta entre usuários como alternativa aos modelos então existentes, predominantemente baseados no paradigma cliente/servidor.

Assim, pesquisadores logo identificaram os principais desafios para essa classe de aplicações: a eficiência na buscas de dados, a formação de uma rede P2P (*overlay*) eficiente (considerando proximidade de rede) e a manutenção dessa rede face à entrada e saída de nós, ambos em um contexto de larga escala.

Com o tempo, surgiram outras aplicações além do compartilhamento de arquivos. Entre elas, pode-se citar conferência de áudio e vídeo, *Instant Messaging* (IM), aplicações colaborativas. Outro forte uso dos *overlays* é a tentativa de montar uma árvore de distribuição de multicast eficiente em nível de aplicação [52], pois boa parte das redes mundiais ainda não habilitaram suporte a multicast nativo.

A implantação de mecanismos de segurança em redes P2P enfrenta uma série de desafios. Embora aplicações P2P possam contribuir para o compartilhamento de recursos e colaboração em larga escala, em ambientes geograficamente distribuídos com controle descentralizado e acoplamento fraco, a sua diversificação e disseminação são dificultadas pela atual falta de segurança.

1.2 Motivação da P2PSL

Ainda é difícil desenvolver aplicações P2P que atendam a múltiplas combinações de aspectos de segurança, a saber, confidencialidade, autenticidade, integridade, autorização, auditoria, não-repúdio, reputação e anonimato. Há quatro razões para tal. Primeiro, os esquemas atuais para segurança de aplicações P2P cobrem apenas aspectos específicos (como por exemplo autenticação e reputação) e não podem ser facilmente integrados em um sistema único.

Segundo, não conseguem isolar os aspectos de segurança da aplicação. Ao invés disso, obrigam o usuário ou desenvolvedor a lidar com uma interface de programação potencialmente complexa, e passar por um penoso processo de configuração.

Terceiro, as abordagens na literatura demandam um comportamento simétrico e uniforme de todos os *peers* que executam uma aplicação. Esta limitação é indesejável em algumas aplicações P2P, quando os requisitos de segurança podem variar significativamente entre usuários. Para ilustrar com um exemplo trivial, tome-se o

Skype, uma aplicação de Voz-sobre-IP: um dado *peer* pode estabelecer diferentes tipos de comunicação com outros *peers*, cada um com seus requisitos de segurança próprios (por exemplo, usuários domésticos e corporativos podem ter diferentes necessidades).

A quarta e última razão é que os esquemas atuais oferecem pouco ou nenhum suporte para implantação gradual, porque eles precisam estar disponíveis em todos os *peers* de uma aplicação. É muito difícil, se não impossível, impor uma adoção abrupta de um novo esquema de segurança em um sistema de larga escala, de acoplamento fraco. Ao invés disso, é importante para o sucesso de sistemas P2P que os mesmos permitam a coexistência entre pares com e sem suporte de um esquema de segurança.

Para atacar esse problema, foi proposta a Peer-to-Peer Security Layer, ou P2PSL [31], uma camada de segurança visando abstrair e desacoplar os conceitos de segurança da comunicação em redes P2P. Ela permite a inclusão de funcionalidades de segurança em aplicações P2P e trata dos problemas de falta de integração, isolamento, assimetria e implantação gradual, recém mencionados. A P2PSL isola a implementação de aspectos de segurança e sua configuração tanto da aplicação P2P como do *middleware* de comunicação subjacente. Cada par pode especificar requisitos de segurança distintos (de acordo com diferentes graus de restrição) para cada canal de comunicação estabelecido com outros pares. Além disso, a implantação da P2PSL pelos pares que compõem a aplicação pode ser feita gradualmente. A implementação está baseada em JXTA, um conjunto consolidado de protocolos para o desenvolvimento de sistemas P2P, que tem sido amplamente usado pela comunidade.

1.3 Objetivos

Este Trabalho de Conclusão tem por objetivo repensar a arquitetura da P2PSL em função de aspectos de escala, transiência de grupos de pares e segurança. Dessa forma, o trabalho visa analisar os vários aspectos da camada de segurança P2PSL expondo seus pontos fortes e suas desvantagens. Essa exposição visa mostrar as fragilidades da camada na tentativa de melhorá-la e propor soluções para os problemas encontrados. A proposta de soluções tende a transformar a P2PSL numa ferramenta de fácil utilização contendo mais alternativas de segurança dos que atualmente ela disponibiliza.

Mais precisamente, esse trabalho visa analisar a P2PSL em nível arquitetural, ou seja, sua composição junto ao substrato. Juntamente com isso, analisar aspectos de escalabilidade e, por fim, analisar as vulnerabilidades de segurança que a P2PSL apresenta segundo os tipos de ataques encontrados na literatura. Também serão apresentadas soluções e alternativas na tentativa de solucionar os problemas encontrados. Por fim, será feita uma prova de conceito no código atual da P2PSL...

[GF2] Comentário: Inserir brevemente quais serão as modificações

1.4 Organização da Monografia

O restante da monografia está organizada conforme listado a seguir. O Capítulo 2 discorre sobre os conceitos de redes Peer-to-Peer, os tipos de aplicações que podem se beneficiar delas, descreve os modelos de P2P e por fim apresenta basicamente aspectos de segurança para redes P2P. O Capítulo 3 discorre sobre criptografia e gerência de chaves, mostrando os modelos e os problemas decorrentes da troca de chaves. O Capítulo 4 mostra em detalhes a P2PSL explorando seus pontos principais. O Capítulo 5 faz uma análise crítica sobre a arquitetura e escalabilidade da P2PSL e discorre as vulnerabilidades encontradas. O Capítulo 6 apresenta propostas para arquitetura, para amenizar os problemas de escalabilidade e para aumentar a segurança provida pela camada. O Capítulo 7 descreve a implementação feita como prova de conceito. Por

fim, o Capítulo 8 fecha o trabalho mostrando as conclusões e possibilidades de trabalhos futuros.

[GF3] Comentário: Lembrar de modificar a descrição do capítulo 7 dependendo do que foi implementado

2 Conceitos de Redes Peer-to-Peer

Segundo [42], no início da Internet, entre os anos 60 e 70, o conceito de *Peer-to-Peer* (P2P) era o predominante como modelo de comunicação, pois não existia controle restrito como acontece hoje nos servidores. A Internet era baseada na prestação de serviço entre máquinas e não existia controle de acesso, pois as redes eram controladas e todos os nós da rede agiam tanto como clientes quanto como servidores trazendo a tona essa característica P2P.

Com a popularização da Internet e a necessidade de mais controle sobre aplicações e usuários, o modelo cliente/servidor se torna predominante. O modelo cliente/servidor já era predominante em Intranets onde o sistema era todo controlado por *mainframes* e os clientes eram máquinas em uma rede local. Esse modelo está representado na Figura 1 onde um servidor opera em uma rede local para servir os clientes sem comunicação com outros servidores, ou seja, os sistemas eram homogêneos e não havia muita interoperabilidade.

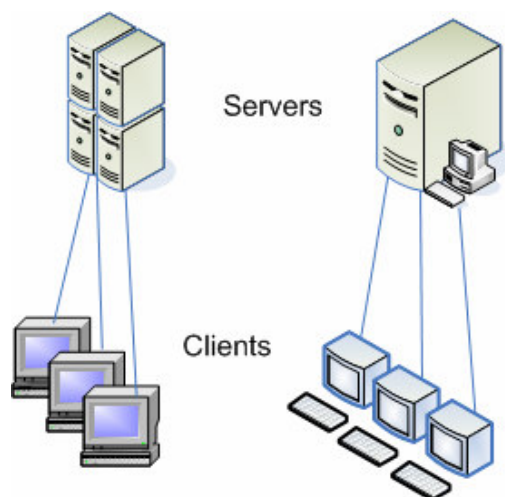


Figura 1: Modelo cliente/servidor usando Mainframes

Com o passar do tempo e o surgimento da internet e a popularização da *web* [41], o paradigma de cliente/servidor se tornou ainda mais popular e amplamente utilizado por provedores de serviço e informação. Desta vez, são utilizados protocolos de domínio público que provêm interoperabilidade entre sistemas heterogêneos. O protocolo mais utilizado hoje para essa intercomunicação é o HTTP. A Figura 2 representa esse modelo de cliente/servidor utilizando a Internet.

Na década de 90, o conceito de P2P volta a ser explorado em larga escala principalmente pela aplicação Napster [39] que permitia o compartilhamento de arquivos de música (MP3). A Figura 3 representa uma rede P2P onde os nós têm contato direto uns com os outros sem a intervenção de um servidor central. Essa abordagem faz com que os nós das bordas da Internet participem mais ativamente na distribuição de informação. As conexões entre nós ocorrem principalmente pela Internet, mas podem ocorrer também em redes locais, redes privadas ou redes *wireless*.

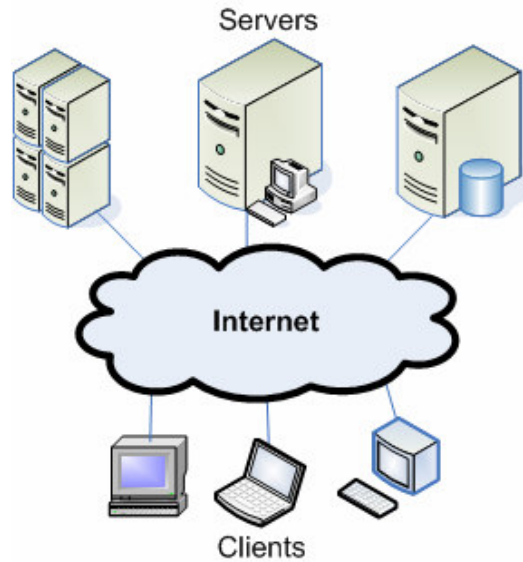


Figura 2: Modelo cliente/servidor na Internet

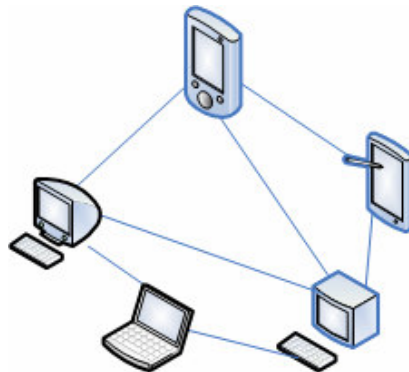


Figura 3: Rede Peer-to-Peer

As redes P2P diferem em vários aspectos do modelo tradicional cliente/servidor. Conceitualmente, a sua principal diferença está na ausência de um servidor central, porém algumas redes P2P apresentam servidor para fazer a inicialização (bootstrap) e realizar algumas tarefas de controle.

Na literatura não existe consenso na conceitualização de uma rede P2P. Segundo [11] existem caracterizações que consideram redes P2P apenas aquelas cujos nós são completamente equivalentes em termos de funcionalidade e das tarefas que eles executam. Porém essa definição falha ao não descrever redes que utilizam o conceito de Supernós e as que utilizam servidores para algumas tarefas como inicialização de nós na rede. Em [12] encontra-se uma definição parecida, pois define P2P como sendo uma rede sem organização hierárquica e sem controle centralizado.

Em [13] é proposta uma caracterização com maior aceitação. Então sistemas P2P são definidos como “uma classe de aplicações que tira vantagens de recursos – armazenamento, ciclos de CPU, conteúdo, presença humana – disponível nas bordas da internet”. Porém, essa definição engloba algumas redes que não são P2P como as que utilizam apenas os conceitos de bag-of-tasks. Com isso, percebe-se que não existe uma definição precisa sobre o que é ou não uma rede P2P.

Em [14] são descritos os requisitos que uma rede P2P deve suportar:

- possibilidade de incorporação de nós localizados nas bordas das redes;
- suporte à conectividade variável ou temporária dos nós, bem como a utilização de endereços variáveis;
- capacidade de lidar com diferentes taxas de transmissão entre nós;
- autonomia parcial ou total dos nós em relação a um servidor centralizado;
- escalabilidade;
- possibilidade de comunicação direta entre nós.

O modelo P2P é atrativo, pois agrega uma série de funcionalidades ao sistema. Um aspecto importante neste modelo é a inexistência de servidor único que configura um ponto central de falhas, dessa forma o sistema se torna mais robusto contra falhas e ataques externos. Se um nó é atacado, somente este nó será prejudicado, mas o restante do sistema continuará respondendo. Outra funcionalidade importante é a capacidade de compartilhamento de recursos dos nós conectados na rede, que fornecem de forma transparente suas capacidades.

Adaptabilidade à transiência de nós na rede faz com que o sistema continue a prover as suas funcionalidades mesmo com alta taxa de entrada e saída de nós (*churn*). Essa adaptabilidade torna o sistema estável e disponível para os usuários conectados contribuindo para auto-organização da rede. Segundo [15], a distribuição da responsabilidade de fornecer serviços por todos os pontos da rede é uma das grandes vantagens das redes P2P.

Para formação de uma rede P2P um overlay é criado. Um overlay é uma rede de sobreposição que é criada sobre a rede de IP tradicional. O overlay é utilizado para prover abstração para a camada de comunicação, pois esta considera somente a conectividade 'virtual' dos nós para contagem de *hops* e não como estes estão ligados à rede física. Contrastando com essa abstração, foi proposta [16] uma arquitetura que tenta fazer as ligações entre os nós do overlay de maneira mais próxima possível da rede física real, dessa forma evita-se que informações que deveriam percorrer apenas um *hop* no overlay acabem percorrendo múltiplos *hops* na rede real.

A seguir, são apresentadas as principais classes de aplicações para redes P2P e então os modelos de organização: estruturada e não-estruturada.

2.1 Aplicações Peer-to-Peer

Para melhor contextualização, serão representados abaixo os tipos de aplicações P2P conhecidos na literatura:

- **Compartilhamento de arquivos:** É o sistema mais disseminado atualmente, pois permite que qualquer usuário da rede possa publicar arquivos (conteúdo) de forma bastante simples. Usuários também podem fazer buscas e obter arquivos sem qualquer restrição. Dessa forma, a rede funciona como um grande sistema de distribuição. Tipicamente o conteúdo dos arquivos publicados é imutável, fazendo que com o tempo ele se torne mais 'popular' e mais disponível na rede. Exemplos desses são: Napster [17], Gnutella [18], KaZaA [19], BitTorrent [20] e DirectConnect [21];
- **Comunicação entre usuários:** essa categoria de aplicações está caracterizada como trocam mensagens entre duas ou mais pessoas na rede. Essas mensagens podem ser de desde texto até vídeos (passando por imagens e voz). Isso ocorre

sem a necessidade de um servidor central, mas algumas das aplicações dadas como exemplo utilizam servidores para autenticação de usuários e outras tarefas de controle. Essas aplicações permitem criação de conferência para múltiplos usuários. Exemplos de aplicações são: MSN Messenger [22], ICQ [23] e Skype [24];

- **Computação distribuída:** classe de aplicações que visam utilizar a capacidade de processamento dos nós conectados na rede. Geralmente, essas utilizam o tempo ocioso dos processadores das máquinas conectadas para computar tarefas que necessitam grande capacidade de processamento. Em computação distribuída, o modelo *bag-of-tasks* é o mais simples e oferece muitos atrativos, pois os dados das tarefas são divididos de maneira independente e podem ser processados em qualquer nó independentemente. Para utilização do conceito de P2P para computação distribuída tem-se a necessidade de comunicação entre os nós para troca de informação para computar dados colaborativamente. Assim, o conceito de *bag-of-tasks* seria muito simplista. Pode-se utilizar outros modelos de computação paralela, porém com alta transiência de nós e interconexões de alta latência dificultam o controle e podem tornar alguns algoritmos ineficientes. Como exemplo, pode-se citar o OurGrid [25];
- **Armazenamento distribuído:** essa abordagem é análoga ao sistema de arquivos NFS [26], mas ao invés dos dados estarem armazenados em apenas um servidor, os mesmos estão distribuídos entre os nós e, portanto precisam de controle de acesso. Se houver replicação é necessário fazer controle de consistência entre as réplicas. Deste modelo, podemos citar o OceanStore [27] como implementação de armazenamento distribuído;
- **Jogos on-line:** essa classe de aplicação tem grande potencial na área de P2P pela escalabilidade, porém praticamente inexistem jogos que utilizam essa abordagem. Os grandes produtores de jogos preferem optar pelo modelo tradicional cliente/servidor (de implementação mais simples) e ter um maior controle do jogo e dos usuários, evitando assim, fraudes e trapaças. Entretanto, na área de pesquisa existem tentativas de criação de jogos utilizando redes P2P como em [28]. Nesse cenário, P2P seria uma escolha natural para jogos massivamente paralelos.

2.2 Redes Não-Estruturadas

Nesse tipo de rede, a estrutura de conexão e organização dos nós é dada de maneira quase randômica, ou seja, quando um nó entra em uma rede já formada, não existe uma posição pré-determinada para ele. Assim, as redes podem ser desbalanceadas, pois nós podem conectar-se, em grande parte, em um grupo pequeno de nós seguindo uma Lei de Potência e por fim, não temos nenhuma pista sobre onde um nó ou objeto pode se encontrar na rede.

Porém, essas redes têm a vantagem de serem de simples implementação e fácil controle para buscas e disseminação e, por esse motivo, tornaram-se as mais populares. Elas facilitam a difusão de arquivos de grandes tamanhos (vídeos, programas, jogos) que ficam muito difíceis de distribuir nas redes estruturadas quando essas usam algoritmos que escolhem onde um objeto vai ser guardado ou quando utilizam replicação. Mas as desvantagens dessas redes são a falta de escalabilidade dos mecanismos de busca, pois em alguns casos elas recorrem à inundação da rede (alta utilização de banda) ou

consulta a servidores (gargalos de comunicação). A seguir, será descrita uma divisão para as redes não-estruturadas.

Centralizada

Um exemplo dessa estrutura é o Napster [17], que utiliza um servidor central que faz todas as tarefas de gerência, consulta e conexão. Quando um novo nó entra na rede P2P, ele conecta-se com o servidor e passa para o mesmo a sua lista de arquivos. O servidor então armazena e indexa essa lista de forma que todas as consultas dos nós são feitas para o servidor e respondidas por ele. O servidor indica para o nó que fez a pesquisa quais os nós que contém a informação desejada. Dessa forma, o nó que deseja a informação tenta se conectar com os nós que tem a informação diretamente para a troca de arquivos. A Figura 4 ilustra esse esquema.

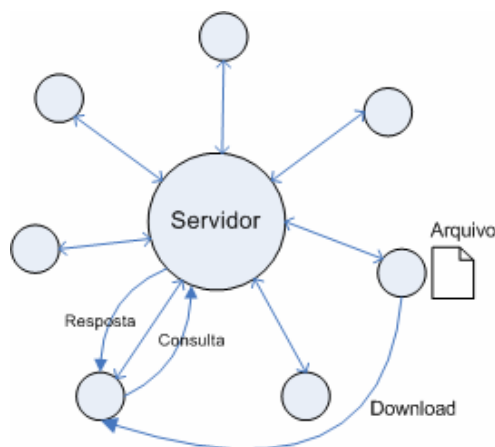


Figura 4: Topologia Centralizada

As desvantagens desse modelo aparecem constantemente na literatura, pois como o mesmo opta pela utilização de um servidor central, este aparece como ponto central de falha no sistema. Além disso, o servidor se torna um gargalo dificultando a escalabilidade e se torna um ponto da rede que pode ser atacado prejudicando toda a funcionalidade da aplicação utilizando essa solução. A centralização facilita a censura, pois como se trata de um servidor fixo, o mesmo é de responsabilidade de uma entidade que pode ser responsabilizada judicialmente pelo conteúdo distribuído como aconteceu com o próprio Napster quando foi processado por inúmeras gravadoras pela distribuição de músicas piratas.

Parcialmente Centralizada

Como exemplo dessa estrutura, podemos citar o KaZaA [19], que utiliza o conceito de Supernós. Um Supernó é um nó pertencente à rede, mas que tem funções equivalentes a de um servidor da estrutura centralizada. Nessa rede, temos alguns nós que se tornam Supernós, então temos uma rede com hierarquia. Nós ditos "comuns" irão apenas contactar o Supernó ao qual se conectaram. O Supernó manterá a lista de arquivos dos nós conectados a ele. Quando um nó dispara uma consulta para um Supernó, este irá verificar localmente em sua base se existe o dado requisitado e depois verificará com outros Supernós se estes conhecem alguma entidade com o conteúdo requisitado. Depois desse passo, o nó que requisitou a consulta, será informado dos nós que dispõem o conteúdo desejado.

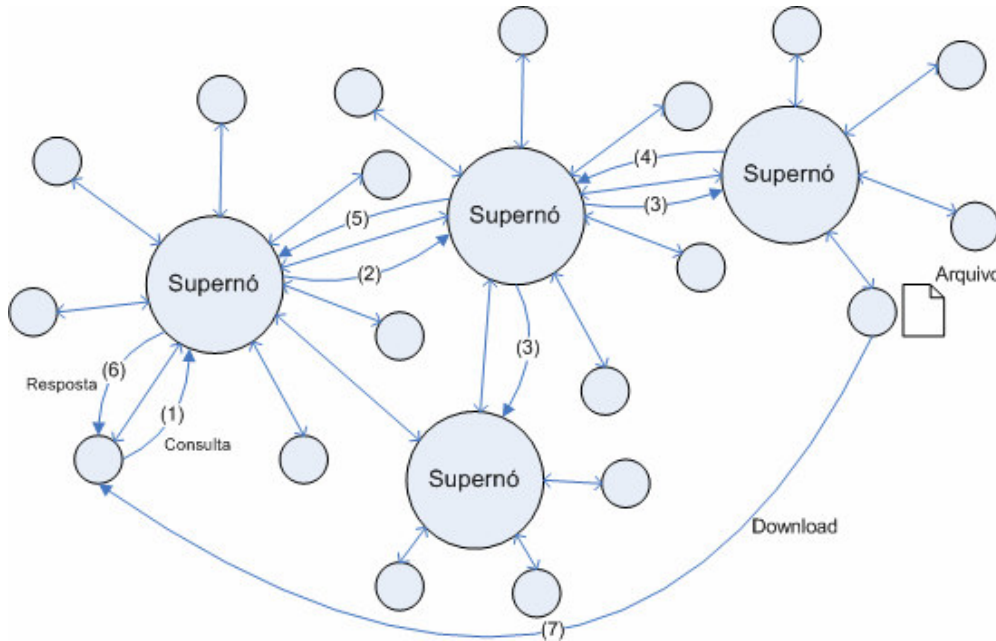


Figura 5: Topologia Parcialmente Centralizada

Essa abordagem é mais escalável que na estrutura centralizada, pois faz uma divisão das tarefas de consulta. A consulta não fica a cargo de apenas um servidor central que necessita de muita banda e muito processamento, mas de um conjunto de servidores que são nós da rede que se dispõem a essa tarefa. A censura nessa rede torna-se mais complexa, porém os ataques de negação de serviço (DoS) podem se focar nos Supernós. Mesmo para ataques de DoS, essas redes conseguiriam manter seu serviço pelo menos parcialmente, pois seria muito difícil conseguir atacar todos os Supernós simultaneamente.

Descentralizada

O representante mais conhecido desse tipo de rede é o Gnutella [18]. Ele é totalmente descentralizado e utiliza apenas uma lista com alguns *hosts*, que podem ser encontrados em gnutellahosts.com, para fazer o *bootstrap* na rede. A partir desse momento, qualquer consulta feita por um nó é requisitada a todos os seus vizinhos. Cada vizinho que recebe uma consulta passa a mesma para todos os seus vizinhos conectados menos o nó que originou a mensagem. Esse algoritmo é conhecido como inundação e se utiliza de TTL (*Time To Live*) nas mensagens para garantir que a mesma não se propague para toda a rede.

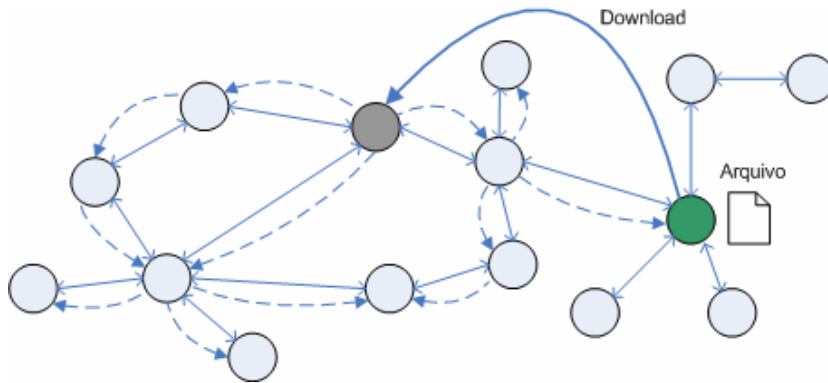


Figura 6: Topologia Descentralizada

Essa abordagem evita boa parte das tentativas de censura e dificulta alguns ataques, porém o sistema consome bastante recursos de rede propagando mensagens. Se o TTL for muito grande, toda a rede será inundada consumindo recursos por quase toda a rede. Se as informações requisitadas por um nó estiverem muito distantes do mesmo e o TTL for muito baixo para não consumir tantos recursos de rede, as requisições podem não alcançar o nó detentor do conteúdo desejado.

Observa-se a necessidade de nós implementarem um sistema de cache para mensagens recentes, pois se uma mesma mensagem chegar repetidas vezes por conexões diferentes, essa somente seja encaminhada uma vez evitando *loops*. Por fim, algoritmos de *random-walks* podem ser usados para diminuir consideravelmente a quantidade de tráfego e ainda conseguir qualidade de consulta aceitável. *Random-walks* é um algoritmo que percorre um caminho na malha de nós aleatoriamente.

2.3 Redes Estruturadas

Nesse tipo de rede os nós são organizados em um grafo onde objetos são mapeados para chaves através de funções *hash* e cada nó da rede fica responsável por um subconjunto do total global de chaves. Entretanto, essa estrutura não permite a busca através de consultas complexas, pois os armazenamentos e consultas são feitos através das chaves e não através do conteúdo. Com isso, conteúdos similares podem estar em lugares totalmente diferentes na rede, pois tem chaves diferentes.

Diferentemente das redes não-estruturadas, nesse modelo um novo nó na rede tem um lugar de conexão no grafo de nós determinado por um algoritmo de *bootstrap*. Como essas redes adotam abordagens totalmente descentralizadas em seu funcionamento, não existe a divisão de arquitetura ilustrada na seção anterior. Logo, para exemplificar esse tipo de abordagem, duas redes que implementam esse tipo de estrutura serão explicadas a seguir para ilustrar o funcionamento das redes estruturadas.

CAN (Content Addressable Network) [29]

Essa rede utiliza um espaço d-dimensional para mapear nodos e chaves. Quando o primeiro nó entra na rede, ele é responsável por todo o espaço, mas à medida que mais nós vão entrando na rede, esse espaço vai sendo particionado entre os nós de forma a fazer balanceamento de carga. Cada área que um nó é responsável é chamada de zona e nesta os objetos são mapeados. O mapeamento ocorre da seguinte forma: Um objeto com valor V é mapeado para uma chave K ; a chave K é mapeada para um ponto P no espaço d-dimensional; essa zona em que o ponto P foi mapeado é de responsabilidade de um nó N ; Assim, o nó N fica responsável pelo objeto de valor V .

Para fazer consultas na rede tem-se um procedimento com passos semelhantes à inserção: um nó N1 fornece para a rede a chave K para recuperar o objeto; essa chave é mapeada para um ponto P na rede; esse ponto P fica na zona Z que é de responsabilidade do nó N2; N1 envia uma mensagem em direção a N2 requisitando o objeto, mas como N1 não sabe o endereço de N2, N1 envia para uma região vizinha que irá passar essa mensagem de região para região até chegar ao nó N2 responsável pela região Z. A Figura 7 mostra o roteamento de uma mensagem enviada por E para C. A mensagem percorre as zonas vizinhas que estão sob controle de A e D antes de chegar até C.

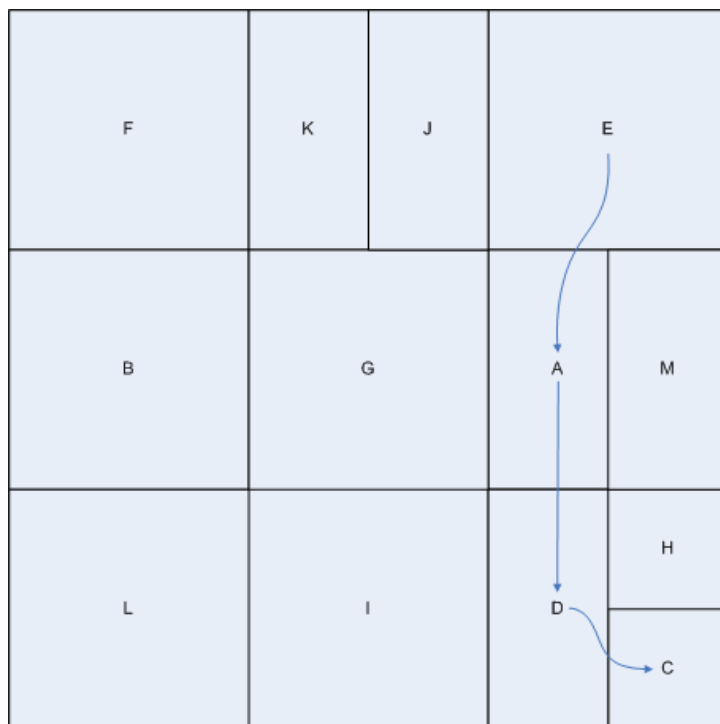


Figura 7: Rede CAN

Chord [30]

Essa rede utiliza um espaço circular para fazer o mapeamento de nós e objetos. O posicionamento de nós e objetos no anel são dados por uma função hash consistente e então ordenados fazendo modulo 2^m (que indica o tamanho do anel). Cada nó que entra na rede é conectado no anel mapeado através da função *hash*. Esse nó contém ponteiros para seu sucessor e uma tabela chamada “*Finger Table*” - utilizada para melhorar o desempenho não necessitando rotear mensagens por todos os nós do anel - onde cada elemento da tabela é o endereço do nó $n+2^i$ (onde n é o número do próprio nó e i é número da entrada na tabela). Com isso, mensagens podem dar 'saltos' na rede.

Para mapear um elemento sua chave é computada e com isso sabe-se em que posição esse objeto deveria estar no anel. O objeto é roteado até sua posição e se não existe um nó responsável pela posição que o objeto foi mapeado, o nó sucessor fica responsável pelo objeto. Para requisições o funcionamento é análogo, pois com a chave do objeto sabe-se a posição do mesmo na rede. Dessa maneira, uma mensagem é roteada até o nó responsável pelo objeto. A Figura 8 representa a formação de uma rede Chord. A *Finger Table* do nó N8 está sendo mostrada e as setas representam as conexões do nó N8 com os responsáveis pelas áreas representadas na *Finger Table*.

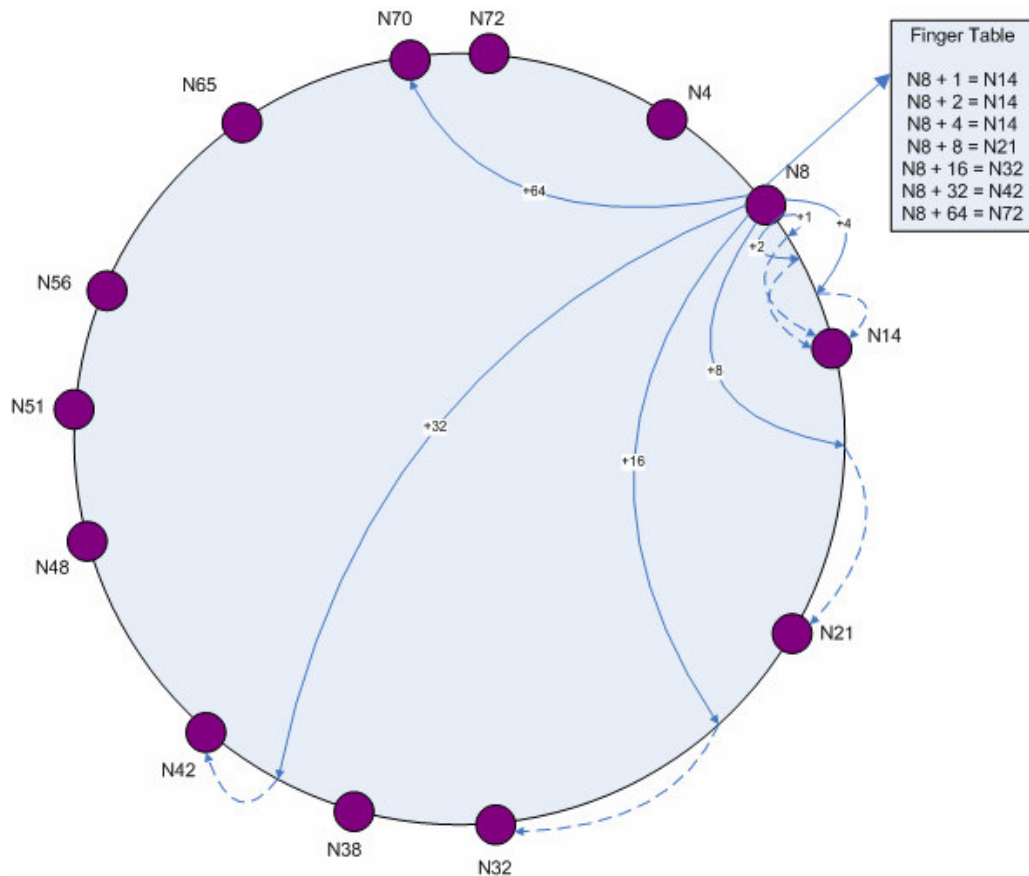


Figura 8: Rede Chord

2.4 Fundamentos de Segurança em Redes Peer-to-Peer

Como visto em [31], a segurança em redes de computadores abrange diversos aspectos que atendem a diferentes objetivos dos pontos de vista dos usuários das aplicações. Esses mesmos conceitos de aspectos de segurança podem ser transportados diretamente para redes P2P ocasionando os seguintes requerimentos:

- **Confidencialidade:** Essa característica, quando aplicada, evita que terceiros consigam identificar a informação trocada entre entidades. Para realizar esse procedimento, duas classes de aplicações são utilizadas: criptografia simétrica e criptografia assimétrica. Do ponto de vista de redes de computadores a criptografia pode ser aplicada na comunicação como um todo ou apenas em alguns campos como dados e cabeçalhos. Em redes P2P, implica que a comunicação entre dois nós da rede não possa ser identificada por outros nós ou por entidades fora da rede.
- **Autenticação:** Requisito que é utilizado quando existe a necessidade de verificar se uma entidade é realmente quem ela afirma ser. Atualmente, para fazer essa verificação utilizam-se assinaturas digitais com algoritmos de criptografia assimétricos, ou seja, um transmissor pode assinar uma mensagem e enviar ao receptor, então o receptor verifica se a assinatura do transmissor está correta. Em P2P isso garante que certo usuário ou nó está sendo identificado e pode-se confiar nas informações provenientes do mesmo.

- **Integridade:** A integridade é a garantia de que uma informação não tenha sido modificada voluntária ou involuntariamente no meio de comunicação. Dessa maneira, se uma mensagem é enviada de um nó A para um nó B, a integridade garante que no caminho de A para B a mensagem não foi modificada. Assim, pode-se saber se houve corrupção da mensagem por fatos isolados como links defeituosos ou se uma entidade maliciosa está tentando injetar informações errôneas na rede ou mensagens. Para garantir integridade é utilizado um algoritmo de *Hash*.
- **Não-Repúdio:** Garante que uma entidade não possa negar o envio de informações feitas por ela. Essa característica já está, em parte, presente quando utilizamos autenticação, pois garantimos que uma entidade é quem ela realmente diz ser. Em redes P2P, isso pode ser utilizado como garantia que certo conteúdo foi enviado por uma entidade. Se esse conteúdo for ilegal ou poluído a entidade que o distribuiu pode pagar o preço de sua ação.
- **Autorização:** Define recursos que uma entidade pode acessar. A autorização faz o controle de acesso de um usuário/nó sobre recursos disponibilizados por outros usuários de forma a limitar o acesso apenas a usuários autorizados. Uma das formas de fazer esse controle é através de *Access Control List (ACL)*.
- **Auditoria:** Capacidade de analisar o comportamento do sistema ou de entidades do sistema. No caso de redes P2P, capacidade de verificar como certo nó está agindo. Para esse tipo de análise é muito utilizado o artifício de *log*. Os *logs* são informações sobre o que está ocorrendo em um nó em um determinado momento (suas ações).
- **Anonimidade:** Permite que uma entidade não possa ser identificada em uma comunicação com outra, ou seja, sua identidade é desconhecida. Se um nó A quer contactar um nó B, sem que B saiba quem o está contactando, A pode utilizar-se de nós intermediários para fazer a comunicação para ele.
- **Reputação:** Indica a confiabilidade de um nó na rede, ou seja, se o mesmo contribui com a rede fornecendo recursos ou apenas onera o sistema sem contribuir com o mesmo. Existem muitos trabalhos sobre esse tema, pois como as políticas dependem de informações dadas por vários nós, o sistema tem uma grande quantidade de vulnerabilidades. De maneira geral, um nó tem melhor reputação se compartilha e fornece recursos e o faz de maneira correta, enquanto terá uma reputação ruim caso contrário.
- **Disponibilidade:** Em sistemas computacionais, essa propriedade refere-se ao fato do sistema estar pronto para responder a requisições de clientes quando for solicitado. Em redes P2P significa que os serviços da rede continuarão disponíveis, mesmo com a alta transiência de nós.
- **Negabilidade:** Permite a uma entidade negar a responsabilidade sobre dados por ela armazenada. Isso ocorre quando uma entidade A armazena dados de outra entidade B, mas esses dados estão cifrados no meio de armazenamento de A, ou seja, a entidade A não consegue identificar os dados armazenados. Isso tem por objetivo garantir que uma entidade não seja processada por armazenar conteúdos com *copyright* que não pertencem a esta entidade.

Tendo em vista os requisitos acima citados, o próximo capítulo irá explorar questões ligadas à criptografia e gerência e troca de chaves entre nós.

3 Criptografia e Gerência de Chaves

Este capítulo aborda os requisitos e garantias providas pelos sistemas criptográficos explorando seu uso e suas fraquezas. Serão apresentados os modelos de criptografia mais conhecidos na literatura e como utilizá-los para prover segurança em sistemas computacionais. Juntamente com isso, o problema de troca e gerência de chaves será abordado tendo em vista a necessidade de utilização do mesmo na P2PSL. Esse capítulo expõe os pontos importantes de sistemas criptográficos e servirá como base para a apresentação de vulnerabilidades e limitações da P2PSL que serão exploradas no Capítulo 5 . As informações contidas nesse capítulo serão usadas para justificar a escolha do modelo de segurança sugerido no Capítulo 6 . Este capítulo está dividido da seguinte maneira: primeiro será descrito os dois modelos tradicionais de criptografia e por fim o problema de troca de chaves será explorado.

Para garantir os requisitos de segurança confiabilidade, autenticidade, negabilidade, integridade, não-repúdio e reputação descritos na Seção 2.4 é necessário empregar mecanismos de criptografia. A criptografia consiste em uma função de embaralhamento de dados que os modifica de maneira que esses não possam ser lidos por entidades que não conheçam o segredo. Algoritmos de criptografia podem ser divididos em duas grandes classes [1], [3], [4], [5], [6] e [9]:

- Criptografia simétrica;
- Criptografia assimétrica.

Cada modelo tem características próprias e é capaz de fornecer diferentes requisitos de segurança. As duas classes baseiam-se na utilização de chaves para cifrar/decifrar o conteúdo de uma mensagem. A seguir, os dois modelos serão expostos mostrando as vantagens e desvantagens de cada um e os problemas decorrentes da gerência de chaves. Essa exploração será utilizada para justificar a escolha do modelo de segurança. No texto as palavras cifrar, embaralhar e criptografar serão usadas intercambiavelmente para descrever o processo de transformar a informação pura que pode ser interpretada em informação protegida, ou seja, não pode ser interpretada. Da mesma maneira, decifrar, desembaralhar e descriptografar serão usados intercambiavelmente para descrever o processo contrário.

3.1 Criptografia Simétrica

A Figura 9 representa o modelo convencional de criptografia [3]. A mensagem original é embaralhada para que seu conteúdo não possa ser interpretado. O algoritmo recebe como parâmetros de entrada uma chave secreta e o texto a ser criptografado produzindo em sua saída o texto criptografado.

Uma vez que o texto criptografado é produzido, este pode ser transmitido através de um canal de comunicação considerado inseguro, pois mesmo que uma entidade não autorizada consiga capturar o texto criptografado esta não conseguirá obter a mensagem original.

No destino, o algoritmo de descryptografia recebe como entrada o texto criptografado e a chave secreta. Dessa forma, ele produz em sua saída o texto original que tinha sido escrito pelo remetente.

A segurança do modelo depende de muitos fatores. Primeiramente, o algoritmo deve ser suficientemente robusto para que seja impossível obter o texto original tendo conhecimento apenas do texto criptografado. Segundo, a segurança dos algoritmos simétricos está baseada no fato da chave ser secreta, ou seja, apenas o emissor e receptor tem conhecimento da mesma. Neste caso, o algoritmo não precisa ser secreto. De fato [4], algoritmos de criptografia de conhecimento pública são considerados mais seguros do que algoritmos secretos. Isso acontece, pois quando um algoritmo ganha domínio público, os criptoanalistas e engenheiros começam a testá-lo tentando encontrar suas fraquezas. Assim, o algoritmo pode ser melhorado, novos procedimentos podem ser necessários ou ainda pode-se optar por não utilizar um algoritmo com certas características.

O fato de o algoritmo ser de domínio público é o que o torna utilizável em larga escala. Como não se precisa mantê-lo em segredo, este pode ser facilmente implementado para várias plataformas e sistemas. Com isso, permite-se a criação de circuitos integrados de baixo custo e de alto desempenho para desempenhar as funções criptográficas [3].

Os algoritmos de criptografia e descryptografia são inversos um do outro, ou seja, se aplicados em seqüência tem-se como saída o texto original.

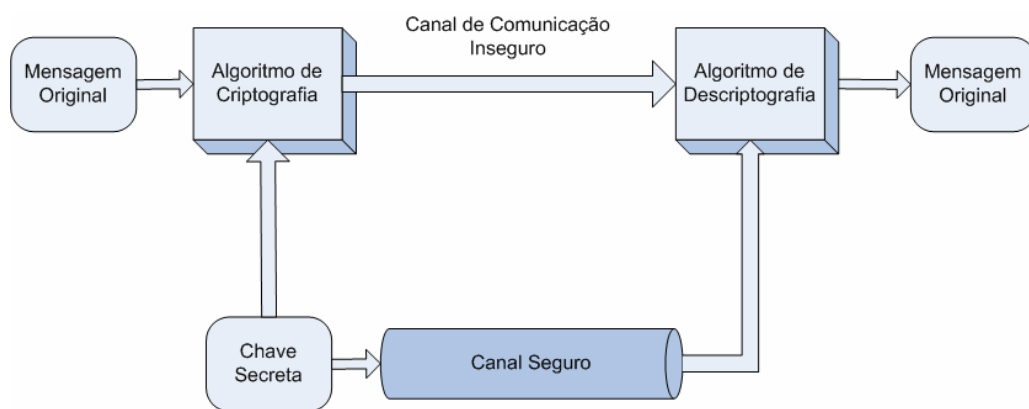


Figura 9: Modelo de Criptografia Simétrica

Para o modelo mostrado na Figura 9, deve-se observar a necessidade de transmissão da chave secreta de forma segura, pois se essa chave for descoberta por uma entidade não autorizada a mensagem original poderá ser obtida e o segredo comprometido. A segurança do modelo está baseada em alguns requerimentos:

- É computacionalmente inviável obter a mensagem original tendo posse somente da mensagem criptografada e do algoritmo;
- É computacionalmente inviável, dado a mensagem original, a mensagem criptografada e o algoritmo, descobrir a chave secreta;
- É computacionalmente fácil, dado uma chave secreta, a mensagem original e o algoritmo, gerar a mensagem criptografada;
- É computacionalmente fácil, dado uma chave secreta, a mensagem criptografada e o algoritmo, gerar a mensagem original novamente.

Muitos algoritmos de criptografia simétrica foram propostos. O primeiro de grande aceitação foi o DES (Data Encryption Standard) que foi adotado em 1977 pelo instituto que hoje é conhecido como NIST (National Institute of Standards and Technology). O DES possui várias fraquezas conhecidas e com isso foram criados outros algoritmos e abordagens na tentativa de criar sistemas robustos contra ataques de força bruta e de criptoanálise. Dentre os algoritmos criados pode-se citar:

- Triplo DES
- IDEA (International Data Encryption Algorithm)
- Blowfish
- RC5
- RC2

3.2 Criptografia Assimétrica

A criptografia assimétrica também conhecida como criptografia por chave pública pode ser considerada uma verdadeira revolução na área da criptografia. Desde o princípio até hoje, as técnicas tradicionais de criptografia eram baseadas em substituição e permutação [3].

A grande revolução na criptografia por chave pública aconteceu porque ela é uma solução puramente matemática e não mais baseada em substituição e permutação. Além disso, essa técnica que, ao contrário da criptografia simétrica, envolve o uso de duas chaves, uma para cifragem e outra para decifragem.

A criptografia assimétrica surgiu da tentativa de resolver dois dos problemas mais difíceis de resolver no campo da criptografia simétrica convencional. O primeiro deles refere-se ao problema da distribuição de chaves, pois quando duas entidades tentassem estabelecer comunicação de forma segura seria necessário que elas tivessem o segredo compartilhado de antemão ou utilizassem um centro de distribuição de chaves.

O segundo problema é o de assinatura digital. Nesse caso, existe a necessidade de termos um tipo de assinatura que tivesse a mesma validade de uma assinatura feita em papel com caneta. Com isso seria possível responsabilizar usuários por mensagens enviadas e termos comércio eletrônico seguro.

Na Figura 10, percebe-se a grande diferença entre o modelo de criptografia simétrica para o modelo de criptografia assimétrica é que no modelo assimétrico não existe a necessidade de um canal considerado seguro para a transmissão das chaves.

Neste modelo, a entidade que deseja receber dados secretamente deve gerar um par de chaves. Nesse par de chaves nomeamos uma das chaves de **chave pública**, pois essa chave será compartilhada e enviada para as entidades interessadas em comunicar-se com a entidade geradora das chaves. A outra chave é nomeada de **chave privada**, pois essa será uma chave que somente a entidade geradora terá conhecimento.

Na Figura 10, pode-se ver que a mensagem original irá ser criptografada usando o algoritmo de criptografia e a chave pública. No destino, a mensagem original será recuperada utilizando o algoritmo de descryptografia e a chave privada. Assim, não é mais necessário compartilhar entre duas entidades uma mesma chave secreta para fazer a comunicação de forma segura, pois a entidade destino irá fornecer publicamente a sua chave pública de forma que qualquer outra entidade possa contactá-la.

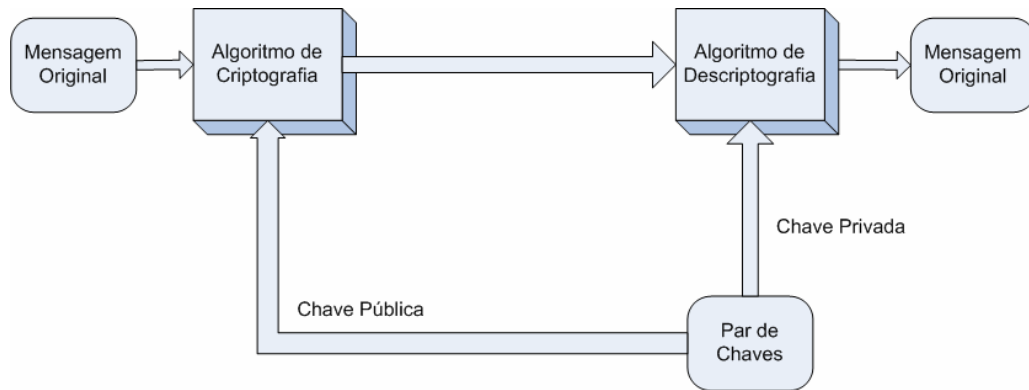


Figura 10: Modelo de Criptografia Assimétrica

Dado a simetria do algoritmo, se ele for aplicado com a chave pública na mensagem original para cifrar os dados, estes poderão ser decifrados usando o algoritmo com a chave privada. Da mesma forma, se o algoritmo for utilizado com a chave privada para criptografar, usa-se o algoritmo com a chave pública para descryptografar.

Os requerimentos para um sistema de criptografia por chave pública podem ser descritos em:

- a) É computacionalmente inviável determinar a chave de descryptografia tendo apenas conhecimento da chave de criptografia;
- b) É computacionalmente inviável determinar a chave privada tendo conhecimento da chave pública, o texto original, o texto criptografado e do algoritmo;
- c) É computacionalmente inviável determinar o texto original tendo posse do algoritmo, do texto criptografado e da chave de criptografia.
- d) É computacionalmente fácil determinar o texto original tendo posse do algoritmo, do texto criptografado e da chave de descryptografia.
- e) É computacionalmente fácil gerar o texto criptografado tendo posse do texto original, do algoritmo e da chave de criptografia;
- f) É computacionalmente fácil gerar o par de chaves.

A aplicação de algoritmos de chave pública é grande e esta é principalmente utilizada nos seguintes cenários:

- **Criptografar/Descryptografar:** Uma mensagem é criptografada usando a chave pública e então transmitida. Posteriormente é usada a chave privada (par da chave pública) para recuperar a mensagem original. Isso já foi ilustrado anteriormente na Figura 10.
- **Assinatura Digital:** Uma entidade assina uma mensagem utilizando o algoritmo de criptografia com a chave privada. Quando a mensagem chega ao destino, este usa a chave pública para recuperar a mensagem original. Como somente uma chave pública (par da chave privada) poderá recuperar a mensagem original, tem-se certeza de que o texto foi gerado pela entidade que detém a propriedade da chave privada que é par da chave pública utilizada para descryptografar.
- **Troca de Chaves:** Chaves de sessão podem ser trocadas utilizando algoritmos de chave pública. Uma determinada entidade gera uma chave de sessão

(podendo ser uma chave para criptografia simétrica) e esta é negociada com outra entidade utilizando as técnicas de criptografia assimétrica.

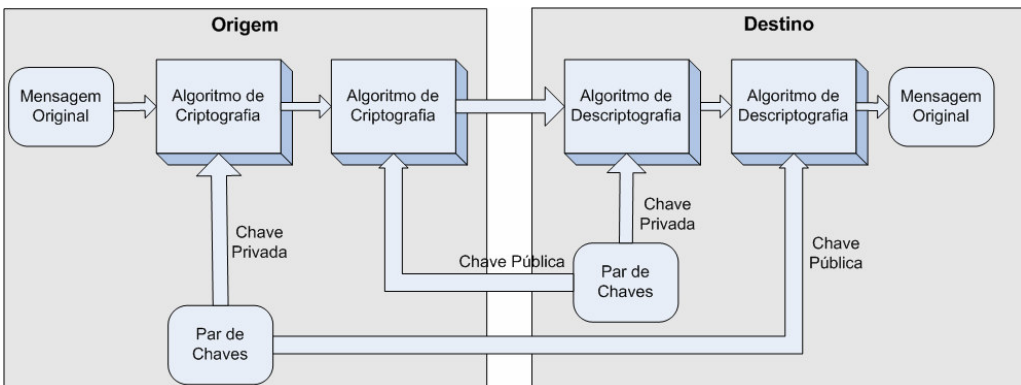


Figura 11: Modelo de Criptografia e Autenticação

O esquema da Figura 11 demonstra a utilização de autenticação e criptografia simultaneamente utilizando algoritmos de chave pública. Primeiramente a mensagem original é criptografada usando a chave privada da entidade origem. Esse processo faz a autenticação, ou seja, tem-se a garantia de que a entidade origem enviou a mensagem, pois somente a chave pública da entidade origem conseguirá descryptografar a mensagem original.

Posteriormente, é aplicado o algoritmo de criptografia com a chave pública do destino. Isso garante a confidencialidade da mensagem sendo transmitida. Aplicando as duas técnicas temos a soma dos requisitos de segurança autenticação, confidencialidade e não-repúdio. No destino, utiliza-se, primeiramente, o algoritmo de descryptografia com a chave privada da entidade destino recuperando os dados autenticados e, após isso, usa-se o algoritmo de descryptografia com a chave pública da entidade origem para checar a origem da mensagem.

Neste grupo de criptografia assimétrica os algoritmos mais conhecidos são [4]:

- RSA
- DH
- DSA
- EC

Os algoritmos RSA e EC podem ser utilizados para criptografia, troca de chaves e assinatura. O algoritmo DSA somente pode ser utilizado para assinatura enquanto o algoritmo DH somente pode ser utilizado para troca de chaves.

3.3 Gerenciamento e Troca de Chaves

O gerenciamento e troca de chaves diz respeito à inicialização do sistema de criptografia, ou seja, o ponto em que uma entidade deve definir quais chaves utilizará para se comunicar com outras entidades.

Os tipos de distribuição de chaves podem ser enumerados da seguinte forma [1],[3],[4] e [5]:

- Distribuição de chaves secretas;

- Distribuição de chaves públicas;
- Uso de criptografia por chave pública para distribuir chaves secretas simétricas.

As subseções a seguir descrevem os esquemas de distribuição de chaves.

3.3.1 Distribuição de Chaves Secretas

Distribuir e gerenciar chaves simétricas sem o uso de criptografia de chave pública é um problema complexo e mesmo tentando proteger esse esquema com procedimentos rígidos de segurança como entregar uma chave pessoalmente, ainda existem muitos problemas inerentes a esse modelo, como o de logística de distribuição. Os modelos de utilização e gerenciamento serão expostos a seguir.

Criptografia por Senha. A criptografia baseada na utilização de senhas segue fielmente o modelo da Figura 9. Uma senha é utilizada como chave para criptografar a informação desejada [4]. Dessa forma, precisa-se garantir que a senha seja transmitida com algum método seguro, ou seja, entregue pessoalmente ou por formas de comunicação que não possam ser comprometidas.

Esse modelo é altamente suscetível a ataques de dicionário e de força bruta, pois um atacante pode utilizar um banco de senhas e com isso ter chaves geradas previamente de forma a apenas aplicar o algoritmo até que uma das palavras do dicionário sirva como chave. Caso o dicionário falhe, um atacante pode continuar fazendo um ataque de força bruta. Como a quantidade de caracteres é baixa, o número de possibilidades de combinações não será muito grande (o menos que frases sejam usadas como senhas, mas isso não é praticável).

Key Encryption Key. A Figura 12 demonstra a utilização dessa técnica. Nessa abordagem, uma chave de sessão aleatória será usada para criptografar os dados e será protegida por um segredo [4]. Os passos para utilizar esse modelo são:

1. Uma chave secreta com tamanho suficientemente grande é gerada aleatoriamente;
2. Essa chave, denominada chave de sessão, é usada para criptografar os dados a serem transmitidos;
3. Uma outra chave denominada KEK (*Key Encryption Key*) será gerada para criptografar a chave de sessão;
4. A mensagem criptografada será enviada juntamente com a chave de sessão criptografada.

O segredo desse sistema está na criação da KEK. Como a chave que foi usada para criptografar os dados é muito forte, pois é aleatória, temos uma forte proteção contra ataques de força bruta aos dados. Com essa dificuldade para ataque direto aos dados, a parte mais sensível que pode ser atacada é a chave de sessão criptografada, ou seja, tentando descobrir a KEK. A KEK será gerada da seguinte maneira:

1. Geram-se dados aleatórios, os quais são chamados de “sal”;
2. Uma senha é escolhida;
3. Mistura-se a senha com o sal para gerar a KEK.

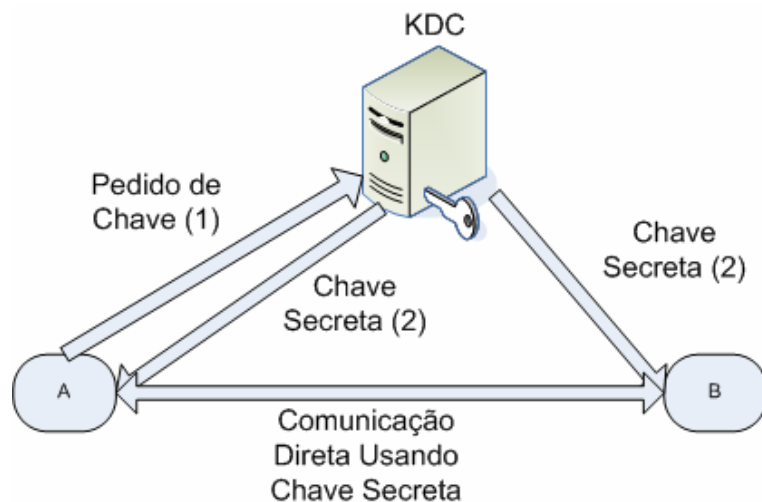


Figura 13: Distribuição de Chave Secreta Usando KDC

Para esse sistema funcionar com confidencialidade, impedindo que um atacante que controla a rede consiga interceptar a chave que está sendo gerada pelo centro, A e B devem possuir chaves secretas únicas para a comunicação com o centro de distribuição. Ou seja, primeiramente teremos A fazendo a requisição para o centro usando sua chave secreta compartilhada com o centro. O centro irá responder com uma chave secreta nova que será criptografada com a chave que o centro compartilha com A. Juntamente com isso, o centro irá compartilhar a mesma chave secreta que foi gerada e enviada para A com B. Essa chave será enviada criptografada com a chave que o centro compartilha com B. Depois que a chave for entregue para as duas entidades, essas poderão comunicar-se usando um algoritmo simétrico e a chave que receberão do centro.

Apesar de parecer interessante, esse esquema tem problemas de logística para distribuição de chaves e impõe uma grande confiança no centro de distribuição. O problema de logística diz respeito à quantidade de chaves que o centro terá de guardar, pois haverá uma chave para cada par de entidades tentando comunicarem-se. O centro ainda deverá ter uma chave guardada para cada entidade que se comunica com ele (pois cada entidade terá uma chave diferente). A chave secreta entre uma entidade e o centro de distribuição deverá ser entregue por um método considerado seguro como telefone ou pessoalmente.

O problema de excesso de confiança diz respeito às informações que o centro tem acesso. Como ele tem a cópia de todas as chaves secretas utilizadas por entidades, será capaz de acessar as informações trocadas por essas entidades ou ainda trair a confiança depositada vendendo as chaves secretas para outras entidades não autorizadas. Ou seja, para entidades que desejam comunicar-se, não existe nenhuma garantia que sua informação não esteja sendo observada.

3.3.2 Distribuição de Chaves Públicas

A criptografia por chave pública apareceu para resolver tanto o problema de distribuição de chaves quanto à autenticidade e não-repúdio de um documento. Dessa forma, distribuir chaves públicas é uma tarefa razoavelmente simples e pode ser feita de várias formas. Os modelos mais comuns de distribuição estão expostos a seguir [1],[3],[4] e [6].

Anúncio Público. Este é o método mais simples, pois envolve apenas a publicação da chave pública de uma entidade diretamente para outra ou disponibilizando a chave pública em qualquer meio sem muito controle como fóruns, repositórios ou servidor próprio. Este método é bastante utilizado pela simplicidade e pelo fato de não precisar envolver nenhum tipo de burocracia ou taxas.

Porém, esse método tem uma grande fraqueza, qualquer pessoa pode forjar a chave pública de outra, ou seja, um usuário pode fazer-se passar por outro. Pode levar muito tempo até a entidade que teve sua chave pública forjada descobrir a farsa e comunicar outras entidades sobre o ocorrido, muita informação pode ter sido revelada.

Diretório Público. Este método é considerado mais seguro que o anterior, pois existe uma entidade que centraliza as chaves. Esse esquema pode ser visualizado na Figura 14 e necessita dos seguintes comportamentos [3]:

- a) O centro mantém um diretório contendo uma entrada do tipo {nome, chave pública} para cada participante;
- b) Cada participante registra a sua chave pública no centro. O registro deve ocorrer pessoalmente ou sob a forma de comunicação autenticada;
- c) Um participante deve ser capaz de repor a sua chave pública com uma nova, ou porque essa chave já foi muito utilizada ou porque descobriu que sua chave privada foi comprometida;
- d) Periodicamente o centro publica as chaves públicas de alguma forma. Por exemplo, pode-se fazer uma cópia física das chaves em um livro como uma lista telefônica ou em algum jornal de grande circulação;
- e) Os participantes podem ter acesso eletrônico com ao centro. Dessa forma, comunicação autenticada e segura é obrigatória.

Mesmo sendo um método mais seguro que o anterior, ainda podem ocorrer alguns problemas. Se a chave privada do centro for comprometida de alguma forma, um atacante pode se fazer passar pelo centro e distribuir chaves públicas falsas de forma a conseguir ler as mensagens enviadas para um usuário. Outra maneira do atacante conseguir isso é atacar o diretório do centro diretamente e modificar sua base de dados. Se entidades que desejam contactar o centro não possuem de antemão sua chave pública, poderão sofrer ataques de man-in-the-middle.

Outros dois ataques ativos podem ser realizados. O primeiro deles se refere a uma entidade tentar fazer-se passar por outra colocando uma chave pública no diretório como se fosse outra entidade, ou seja, forjar identidades. A segunda diz respeito à possibilidade de uma entidade que controla a rede aguardar o envio de uma chave de outra entidade para o diretório. Essa entidade maliciosa descartaria a mensagem e enviaria a sua chave pública para o diretório e enviaria a confirmação do diretório para a entidade requisitante.

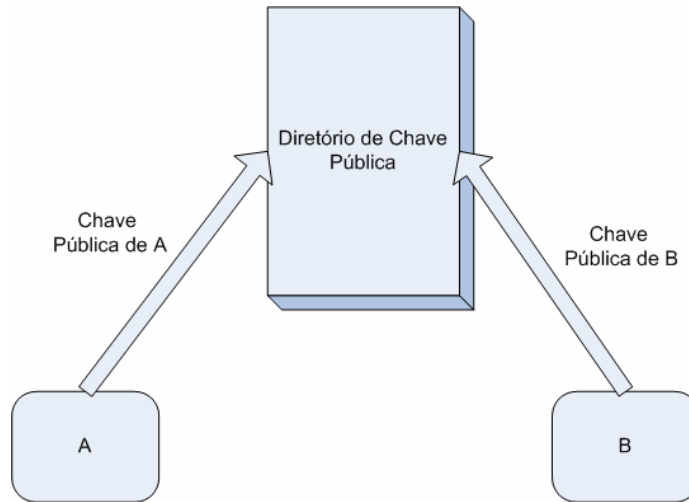


Figura 14: Diretório de Chave Pública

Autoridade de Chave Pública. Uma segurança mais forte pode ser alcançada provendo um maior controle na distribuição de chaves. Na Figura 15 pode-se ver o modelo de funcionamento. Neste cenário, assume-se que os clientes conhecem de uma forma segura a chave pública da autoridade e somente a autoridade tem conhecimento da chave privada correspondente. Os procedimentos ilustrados na Figura 15 são referentes a uma iniciativa de comunicação de A com B onde os dois não se conhecem previamente[3]:

- 1) A envia uma mensagem para a autoridade com uma marca de tempo e uma requisição da chave pública de B;
- 2) A autoridade responde com uma mensagem que é criptografada usando a chave privada da autoridade. Como A tem a chave pública, ele está assegurado que a mensagem partiu da autoridade e não foi modificada. Os elementos da mensagem enviada pela autoridade são:
 - A chave pública de B;
 - A mensagem original enviada por A para que A certifique-se que a mensagem não foi modificada antes de ser recebida pela autoridade;
 - A marca de tempo. Assim A tem a garantia que esta é uma chave recente de B.
3. A guarda a chave pública de B e a utiliza para enviar uma mensagem para B contendo um identificador de A (ID_A) e um identificador de sessão (N_1);
- 4,5. B recupera a chave pública de A da mesma forma que A obtém a chave pública de B.
6. B envia para A uma mensagem criptografada usando a chave pública de A. Essa mensagem contém o identificador de sessão de A (N_1) e um novo identificador de sessão para B (N_2). Como somente B poderia ter descryptografado a primeira mensagem, isso garante para A que seu correspondente é o B (utilizando N_1);
7. A responde N_2 para B usando a chave pública de B. Com isso B garante que o A é o seu correspondente, pois somente A poderia ter descryptografado a mensagem contendo N_2 .

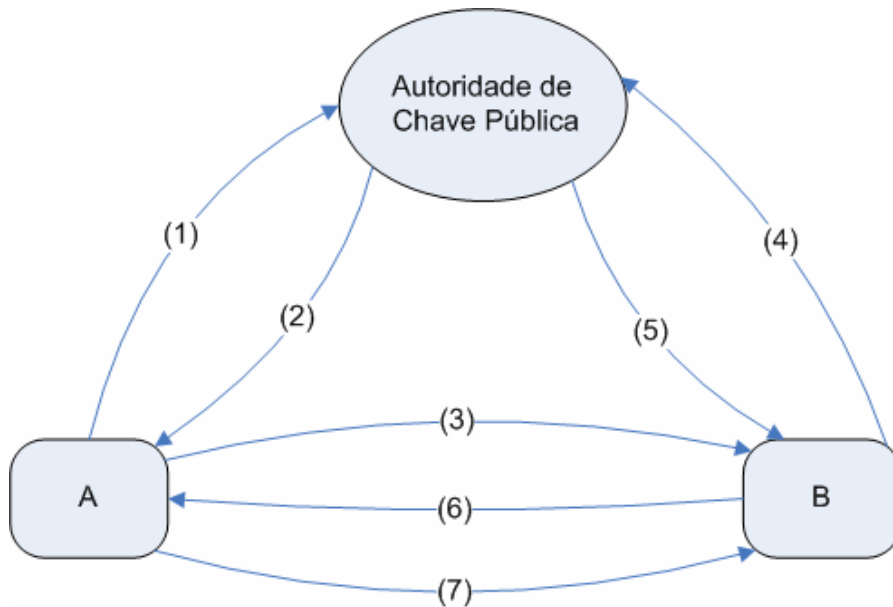


Figura 15: Autoridade de Chave Pública

Esse esquema parece bastante atrativo, porém ele tem algumas desvantagens. A primeira delas é que a autoridade se torna um gargalo para o sistema, pois se tivermos uma quantidade muito elevada de consultas a ela, teremos uma péssima qualidade de serviço, ou seja, não é uma solução escalável. Segundo, esse esquema ainda pode sofrer ataques de modificações das chaves que a autoridade guarda em caso de invasão do servidor.

Autoridades Certificadoras. Este esquema pode ser visto como uma mistura do modelo de autoridade de chave pública e anúncio público. O modelo tem uma semelhança com o anterior, pois possui uma autoridade e é parecido com o anúncio público, pois as partes trocam chaves sem o envolvimento da autoridade. Dessa forma, soma-se o melhor das duas abordagens.

Nesse modelo ([1],[3],[4] e [6]), a autoridade certificadora gera um certificado para uma entidade. Esse certificado irá conter a chave pública da entidade, o nome da entidade, nome da autoridade certificadora, uma validade e outras informações relevantes. Tudo isso, ficará assinado com a chave privada da autoridade. Os requerimentos para esse esquema são os seguintes:

1. Qualquer entidade pode ler o certificado para obter a chave pública e o nome do dono do certificado;
2. Qualquer entidade pode verificar que o certificado foi gerado pela autoridade e não foi forjado;
3. Somente a autoridade certificadora pode criar e atualizar certificados;
4. Qualquer participante pode verificar a validade de um certificado.

A Figura 16 mostra um esquema simplificado e genérico para obtenção de um certificado:

1. Primeiramente, um requisitor deve enviar para a autoridade certificadora o pedido de certificação juntamente com sua chave pública.

2. A autoridade irá responder esse pedido com o certificado. Esse irá conter o nome de A, a chave pública de A e um indicador da validade do certificado. Essas informações estarão criptografadas com a chave privada da autoridade.

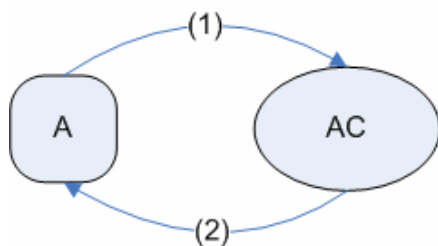


Figura 16: Pedido de Certificação para AC

Cada entidade da rede que deseje fazer uso de certificados para comunicar-se com outras entidades, deve confiar na AC e obter, de forma segura, a chave pública da AC. Desta forma, todas as entidades devem requisitar seu certificado junto a AC. Esse procedimento é feito apenas uma vez por entidade, pois uma vez que uma entidade requisitou o certificado, ela não precisa mais contactar a AC.

De posse do certificado, se uma entidade A deseja contactar a entidade B, ela somente precisa enviar o seu certificado para B. Da mesma forma, B apenas enviará seu certificado para A. Esse procedimento está demonstrado na Figura 17. Assim, as chaves públicas terão sido trocadas com segurança, pois as entidades terão posse da chave pública da autoridade e essa chave será usada para garantir a validade dos certificados garantindo que o mesmo não foi forjado. Posteriormente, as entidades verificam a marca de tempo para verificar se essa ainda é uma chave válida.

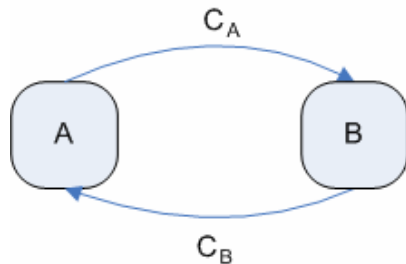


Figura 17: Troca de Certificados

A marca de tempo é utilizada para proteger-se de um oponente que, por algum método como força bruta, invasão ou engenharia social, possa obter a chave privada de um participante. Assim, essa chave expira de tempos em tempos para prevenir esse tipo de acontecimento. Porém, se um usuário descobre que teve a chave privada comprometida, ele deve avisar a autoridade para que essa comunique outras entidades e pessoas que tentem contactar ela que certo certificado foi comprometido e foi anulado.

Os certificados gerados sempre têm um assinante e um dono e os dois serão entidades diferentes. Uma exceção a isso ocorre apenas para o certificado raiz. O certificado raiz é aquele em que a AC assina o seu próprio certificado que é enviado as entidades interessadas.

O padrão X.509 [10] é o mais utilizado para certificação e é considerado o padrão atual. Porém, nesse esquema de segurança ainda existe muita confiança em torno da autoridade certificadora (AC). Essa confiança ainda é menor do que se necessita para

uma KDC, pois a AC não terá acesso às chaves privadas das entidades, impedindo-a de obter as informações que trafegam pela rede.

A emissão de um certificado pode ser de baixa ou de alta confiabilidade [1]. Certificados de baixa confiabilidade são aqueles gerados digitalmente sem a presença em pessoa do representante da entidade envolvida. Para criação do certificado de baixa confiabilidade, geralmente usa-se o e-mail como forma de verificar a entidade que está pedindo pelo certificado, sendo que esse e-mail irá aparecer junto com as outras informações no certificado. Porém, isso não é suficiente para garantir que foi a entidade em questão que fez a requisição do certificado. Para não comprometer-se e especificar para os receptores do certificado, a AC coloca no certificado a informação que este é de baixa confiabilidade.

Quando uma entidade quer gerar um certificado de alta confiabilidade, ela deve contactar a AC pessoalmente ou através de alguma forma de verificação segura apresentando documentos para comprovar sua identidade. Assim, a AC irá colocar no certificado que o mesmo é de alta confiabilidade e a AC garante que o dono do certificado é realmente quem ele diz ser.

Se a chave privada de uma entidade certificada for quebrada, o certificado da autoridade deve ser cancelado imediatamente. Para implementar isso, listas de revogação de certificados (CRL) são utilizadas [4]. A lista é uma estrutura de dados que contém o certificado e a data e hora em que ele foi revogado. Essa lista é assinada pela AC que emitiu os certificados.

Existem duas formas de uma entidade possuir essa lista de revogação atualizada. A primeira maneira é usando o método *pull*, ou seja, de tempos em tempos as entidades consultam a AC ou um repositório onde fique depositada a CRL atual. A segunda forma é utilizando o método *push*, ou seja, a AC transmite as atualizações diretamente para as entidades interessadas.

Pretty Good Privacy. O PGP também utiliza o conceito de certificação e assinaturas digitais, porém não existe o conceito de uma entidade ou autoridade como a AC que controla a distribuição de certificados [1]. O PGP utiliza o conceito de rede de confiança em que uma entidade A confia em um certificado C quando este for assinado por uma entidade B a qual A confia.

Para iniciar o sistema, cada entidade deve gerar seu par de chaves assimétricas e criar o seu próprio certificado assinando-o com a chave privada. O funcionamento é demonstrado na Figura 18. A entidade A envia o certificado para uma entidade B, B deverá encontrar alguma forma de validar esse certificado, no exemplo em questão utiliza-se comunicação via telefone. Para efetuar a validação é possível usar campos como *fingerprint* ou verificar os dígitos da chave pública.

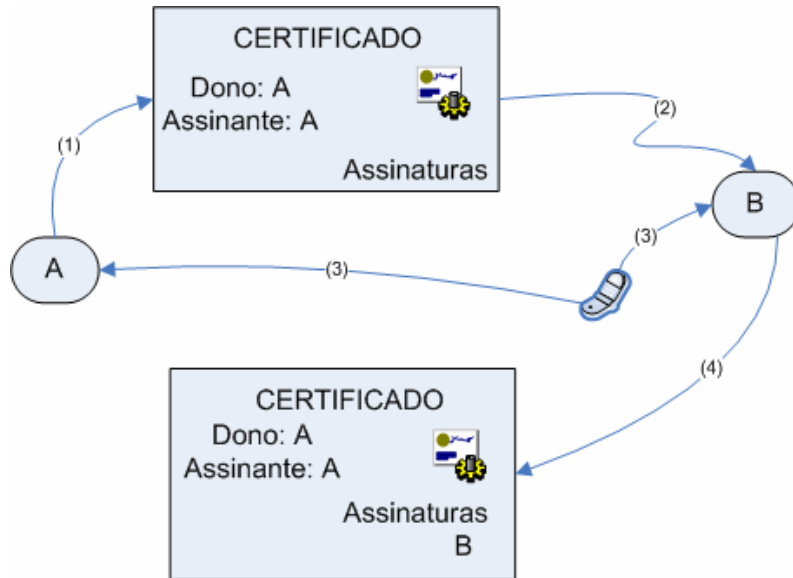


Figura 18: Assinatura do Certificado de Outra Entidade

Depois que a entidade B certificar-se de que o certificado recebido pertence à entidade A, B pode assinar o certificado de A com sua chave privada. Essa assinatura indica que B confia no certificado da entidade A.

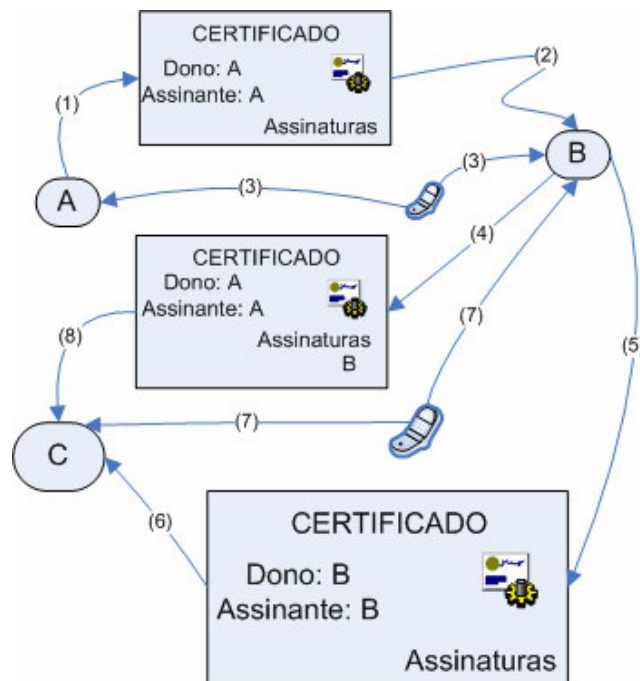


Figura 19: Rede de Confiança

Como B assinou o certificado de A garantindo sua origem, B pode transmitir o certificado assinado por ele de volta para A. Dessa forma, A pode, se uma nova entidade C desejar ter posse da chave pública de A, enviar o certificado assinado. A Figura 19 mostra como C contacta B pedindo a chave de A. Frisa-se que para esse esquema ocorrer é necessário que C confie em B. C pode usar o mesmo esquema representado na Figura 18 para obter a chave pública de B. Posteriormente, B envia o certificado de A

para C e como B já aferiu a veracidade do certificado, C confiará no certificado sem a necessidade de contactar A.

PGP fornece um bom nível de segurança e é, atualmente, bastante utilizado. Esse esquema é uma alternativa a utilização de Autoridades Certificadoras, porém carece de qualquer controle centralizado. Isso faz com que não haja utilização deste modelo para comércio eletrônico, pois existe a necessidade de uma entidade assumir a responsabilidade pela geração e controle dos certificados. Outro grande problema é a possibilidade de entidades entrarem em conluio para enganar uma entidade fora do grupo. De alguma forma, um dos representantes do conluio consegue que uma entidade confie nele e com isso tenta fazer com que essa entidade confie no grupo todo.

3.3.3 Distribuição de Chaves Secretas Usando Criptografia Assimétrica

A utilização de criptografia de chave pública resolve vários dos problemas para comunicação segura e autenticação. Porém, a sua performance é relativamente baixa se comparada à criptografia simétrica. Por esse motivo a criptografia simétrica é utilizada em conjunto com a assimétrica para prover segurança, autenticação e desempenho aceitáveis. Três técnicas para distribuição de chaves secretas serão mostradas a seguir ilustrando como a criptografia assimétrica é utilizada apenas para a troca de uma chave secreta, denominada chave de sessão. Essa chave será posteriormente descartada e toda nova comunicação terá uma nova chave de sessão criada.

Distribuição Simples de Chaves Secretas. Como vemos na Figura 20, temos A e B tentando comunicação mútua. Como A está tentando iniciar a comunicação, teremos os seguintes passos:

1. A cria um par de chaves assimétricas (pública e privada) e envia a chave pública para B;
2. B gera uma chave secreta K e envia para A criptografando-a com a chave pública de A;
3. A recupera a chave secreta K descriptografando a mensagem com sua chave privada;
4. A e B descartam as chaves assimétricas e ficam somente com a chave secreta K.

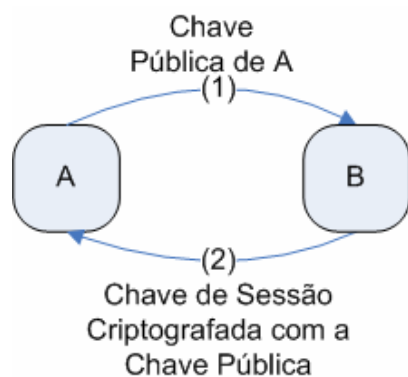


Figura 20: Distribuição Simples de Chave Secreta

Neste momento em diante, A e B podem trocar mensagens utilizando a chave de sessão secreta K, pois apenas eles a conhecem e podem interpretar os dados trocados. Porém, esse sistema apresenta uma falha: pode ser comprometido por ataques ativos.

Um ataque do tipo “man-in-the-middle”[8] poderia acontecer no momento em que A transmitisse a chave pública para B, assim toda a comunicação subsequente estaria comprometida. O atacante trocava a chave pública de A por uma que ele tivesse conhecimento da respectiva privada, dessa forma o atacante teria conhecimento do segredo compartilhado (chave de sessão).

Distribuição de Chave Secreta com Confidencialidade. Essa técnica é parecida com a anterior, porém não usa chaves públicas temporárias. Neste caso, assume-se que as entidades A e B que desejam comunicar-se já possuem a respectiva chave pública da outra entidade. Isso pode ser alcançado usando uma das técnicas anteriormente citadas.

Na Figura 21 o modelo é representado. Como as chaves públicas já são conhecidas, temos a seguinte seqüência de passos:

1. A gera uma chave secreta K e criptografa ela com a chave pública de B;
2. B recebe a mensagem e descriptografa a mesma usando sua chave privada. Dessa forma pode obter a chave secreta;
3. Agora basta as duas entidades comunicarem-se usando a chave secreta K.

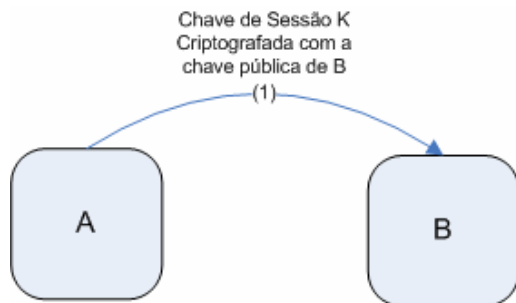


Figura 21: Distribuição de Chave Secreta com Confidencialidade

Esse sistema é mais robusto que o modelo anterior. Porém, também pode sofrer de um ataque “man-in-the-middle”. A forma de ataque aqui seria diferente do anterior, pois um atacante C deveria interceptar a comunicação, descartar a mensagem de A para B com a chave de sessão e gerar uma nova chave de sessão criptografando-a com a chave pública de B. Quando B começasse a enviar dados para A usando a chave de sessão recebida, C poderia verificar os dados da transmissão, porém essa farsa seria descoberta rapidamente, pois A e B não estariam sincronizados com a mesma chave secreta.

Os problemas de intromissão nesse modelo podem ser resolvidos utilizando confirmação de recepção das mensagens com identificadores de sessão, pois esses não teriam como ser forjados, ou seja, um identificador de sessão que foi enviado deve ser o mesmo que é recebido em uma confirmação, caso contrário houve modificação nas mensagens.

Distribuição de Chave Secreta com Confidencialidade e Autenticação. Esse esquema agrega um passo a mais ao esquema anterior. Antes de a mensagem ser criptografada ela é assinada. Como no modelo anterior, este assume que as chaves públicas já foram previamente distribuídas por algum método como os citados anteriormente. A Figura 22 ilustra o funcionamento:

1. A gera uma chave de sessão K assinando-a com sua chave privada e, posteriormente, criptografando-a com a chave pública de B.

2. B descriptografa a mensagem com sua chave privada e posteriormente verifica a autenticidade da chave secreta usando a chave pública de A.
3. Agora a chave de sessão K pode ser usada pelas duas entidades para troca de mensagens usando criptografia simétrica.

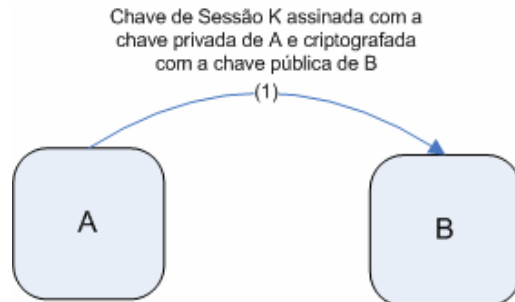


Figura 22: Distribuição de Chave de Sessão usando Autenticidade e Confidencialidade

A assinatura não estará correta e isto configurará uma suspeita de ataque caso aconteça qualquer tentativa de intervenção ativa na transmissão das mensagens. Assim as entidades ficarão sabendo da situação de não confiabilidade da rede.

4 Peer-to-Peer Security Layer

A P2PSL (*Peer-to-Peer Security Layer*) foi proposta em [31] para prover uma camada de segurança modular para integração com aplicações P2P, pois apesar de existir arquiteturas tentando prover escalabilidade e segurança [33], elas ainda são complexas e existe a necessidade de reescrever toda aplicação novamente. A P2PSL tenta sanar dificuldades encontradas na implementação de sistemas P2P seguros ([7] e [32]) utilizando uma implantação gradual.

Como a implantação de aspectos de segurança em aplicações P2P ainda é um problema bastante presente, a P2PSL tenta atingir as seguintes metas [31]:

- Encapsulamento: prover uma camada que implementa requisitos de segurança com uma interface simples de maneira a não inserir complexidade de implementação ao código da aplicação P2P, deixando os aspectos de segurança isolados dos outros módulos.
- Modularidade: a inserção de segurança em sistemas aumenta o custo e processamento. Assim, deve-se permitir ao usuário escolher quais requisitos este deseja implantar sem a necessidade de obrigá-lo a utilizar todos.
- Implantação Gradativa: permitir a implantação em um sistema em utilização sem a necessidade de atualizar todas as aplicações simultaneamente, pois atualizar uma aplicação em todas as bordas gera um esforço muito grande e aplicações não atualizadas deixariam de funcionar. Dessa forma, o sistema antigo poderia coexistir com o novo de maneira transparente.
- Re-configurabilidade: a natureza autônoma das aplicações P2P demanda a configuração sob demanda de requisitos de segurança. Essa característica permite que o sistema possa trocar os requisitos de segurança a qualquer momento.

A P2PSL foi implementada independentemente da aplicação sob a qual ela irá executar [7] e do substrato de rede subjacente. Sua API é de fácil manipulação e permite a inserção gradual de requisitos de segurança em aplicações.

Uma rede P2P tem natureza assimétrica de operações, pois cada nó é independente e pode exigir níveis de segurança que sejam mais adequados para cada momento e situação de um ambiente. Pensando nisso, a P2PSL foi concebida de maneira modular e com configuração individual nos clientes para que as aplicações possam optar por seus requisitos de maneira independente.

4.1 Visão Geral

P2PSL pode ser visualizada como a combinação de vários módulos que, quando combinados, agregam os requisitos de segurança descritos na Seção 2.4. A Figura 23 ilustra uma rede P2P com quatro nós utilizando a P2PSL. Cada nó tem em seu corpo alguns dos módulos para incorporação de segurança e cada um deles define políticas para utilização dos módulos. Para evitar a necessidade de configurar políticas de segurança para cada nó da rede, a camada oferece o conceito de perfis. Tome-se como

exemplo o Peer2 da Figura 23, ele tem dois perfis configurados. O Perfil A requisita autenticação para envio e recebimento de mensagens, o Perfil B requisita autenticação para envio e autenticação mais confidencialidade para recebimento de mensagens.

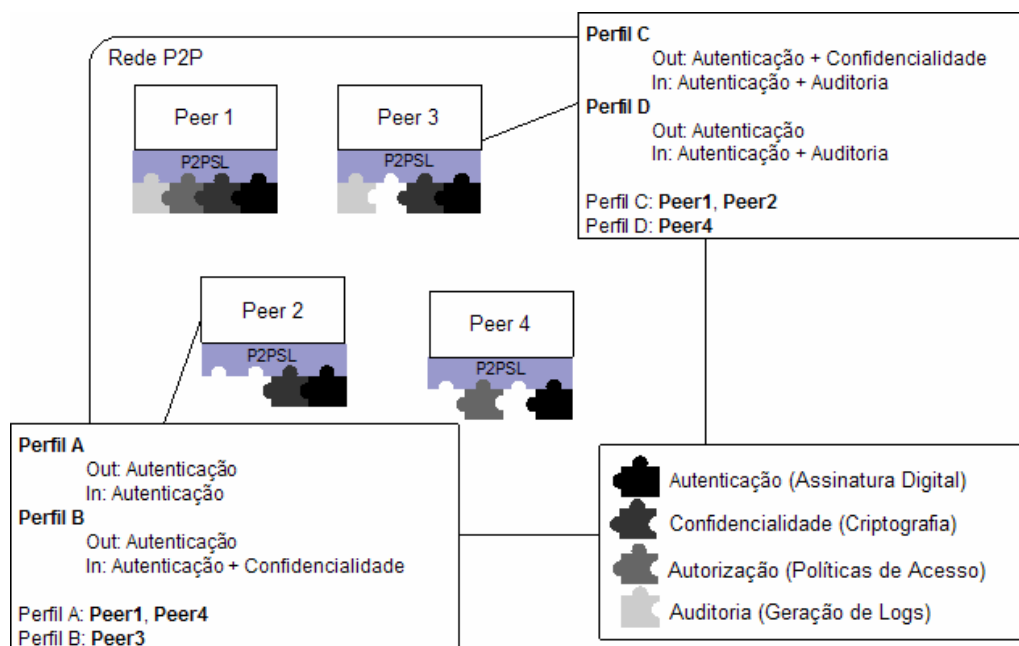


Figura 23: Combinação de Perfis e Aspectos de Segurança nos Nós usando P2PSL

Neste exemplo, os nós Peer1 e Peer2 foram colocados no Perfil A e o nó Peer3 foi colocado no Perfil B. Isso simplifica a configuração, pois podemos atribuir alguns perfis básicos e selecionar os nós que desejamos colocar nesses perfis. Um perfil “*default*” pode ser criado e atribuir a ele todos os novos nós desconhecidos e então um perfil “*legacy*” pode ser criado para os nós que não estão utilizando a camada de segurança. Isso facilita o suporte aos sistemas legados. Dessa forma pode-se migrar gradativamente o sistema para uma solução com incorporação de segurança sem a necessidade de afetar todos os nós de uma só vez.

4.2 Funcionamento Interno

A camada funciona como um *wrapper* entre a aplicação e o substrato de rede. Ela recebe os dados vindos de um nó pela rede, aplica os módulos necessários para recuperar os dados enviados pela aplicação do nó remetente e entrega os dados para a aplicação. Quando a aplicação deseja comunicar-se com outro nó, essa contacta a camada de segurança enviando os dados e o destino. Dessa forma, a camada aplica os módulos necessários para a mensagem obter os aspectos de segurança estabelecidos no perfil podendo ser entregue para o nó destino através do substrato de rede. A Figura 24 ilustra esse procedimento que executa de forma transparente e essa abordagem se faz necessária, pois existem módulos como o de criptografia que necessitam modificar o conteúdo da mensagem para envio, sendo que o outro lado utiliza o mesmo módulo para recuperar os dados originais.

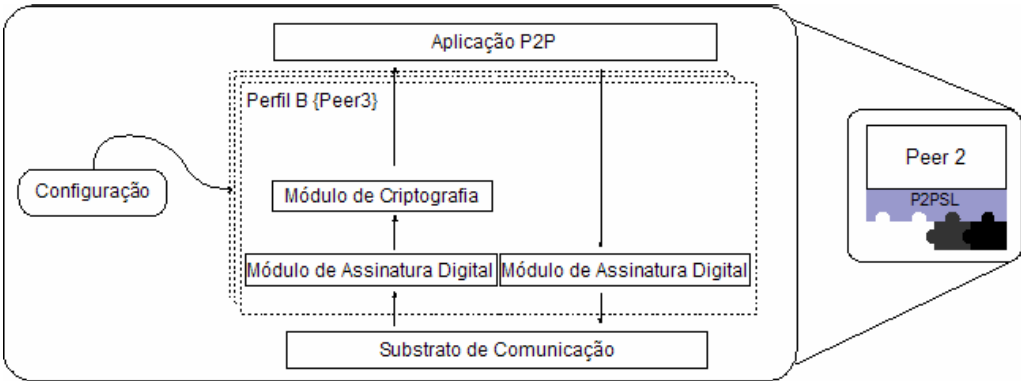


Figura 24: Funcionamento Interno

A Figura 25 mostra como é a implementação e funcionamento da camada dentro de um nó. Como se pode perceber a comunicação é feita baseada em JXTA/JAL permitindo fácil integração da camada com aplicações feitas sobre essa plataforma. Como o substrato é baseado em JXTA, ele terá as garantias oferecidas por essa arquitetura e a P2PSL somente poderá funcionar juntamente com substratos que implementem essa arquitetura.

Internamente, quando uma aplicação deseja enviar dados para outro nó, a aplicação (*JXTA Application*) chama o método *sendMessage()* do *SecurePeer*. Esse método recebe a mensagem a ser transmitida e o destino de maneira que a camada irá, através do arquivo XML que descreve a configuração dos perfis, aplicar os módulos de segurança necessários. Depois que os módulos são aplicados, a mensagem segue para a implementação JXTA/JAL e é enviada pela rede para o nó destino.

Quando a mensagem chega ao nó destino, ela será recuperada pelo *SecurePeer* utilizando o método *receiveMessage()*. Logo após, o *SecurePeer* irá aplicar os módulos necessários definidos no arquivo XML para recuperar a mensagem e então a mensagem é entregue para aplicação através do método *receiveMessage()*.

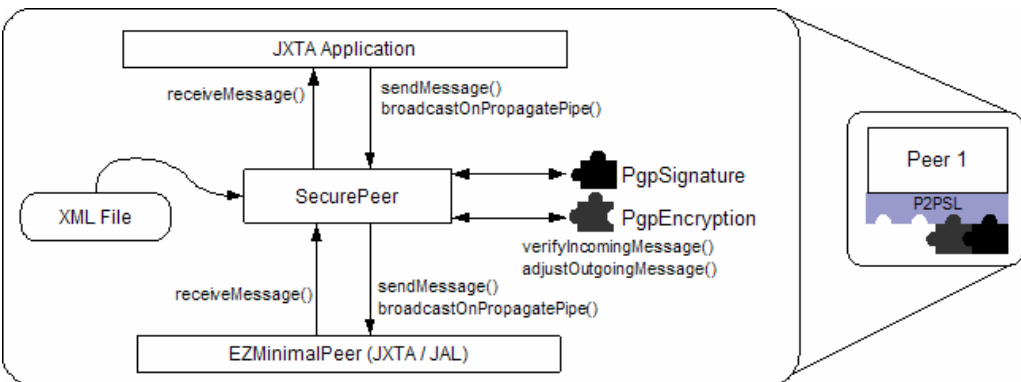


Figura 25: Funcionamento Interno

4.3 Configuração dos Módulos e Requisitos de Segurança

Com sua implementação modular utilizando carga dinâmica de módulos, fica simples a adição de novos módulos de segurança na ferramenta de maneira a atender aos requisitos das aplicações. Para disponibilizar um novo módulo para a P2PSL é necessário apenas estender uma classe abstrata implementando suas interfaces de entrada e saída definindo o comportamento de modificação das mensagens que passam

pelo módulo. Depois disso, basta colocá-lo junto com os outros módulos na estrutura de classes.

Cada módulo deve ser caracterizado de forma a estabelecer suas necessidades e isso é realizado através de um arquivo XML que defini as informações para a camada controlar o comportamento do módulo. Apenas quatro parâmetros precisam ser informados para a camada no arquivo XML, sendo que outros parâmetros podem ser inseridos para o funcionamento do módulo. Como exemplo de parâmetros para o módulo, tem-se a chave pública e privada utilizadas no módulo de criptografia ou o arquivo para gravar as informações do módulo de *logs*. A Figura 26 ilustra um exemplo de configuração de módulo. As informações contidas dentro da *tag* <parameters> são de utilização exclusiva do módulo e não são utilizadas pela P2PSL.

```
<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="xxx" remote="true"/>  
    <parameter name="pass_phrase" default="password"/>  
  </parameters>  
</module>
```

Figura 26: Configuração do Módulo

Para cada módulo desenvolvido, P2PSL associa quatro características que são utilizadas para indicar o comportamento do módulo:

- **export_requirement:** indica se o nó remetente da mensagem necessita modificar a mensagem sendo enviada a outro nó para que este possa aplicar o mesmo módulo e recuperar o conteúdo original da mensagem.
- **obligatory_if_applied:** indica se o nó receptor de uma mensagem necessita utilizar o módulo para recuperar a conteúdo original (que fora modificado pelo remetente utilizando *export_requirement*).
- **allow_on_bcast_sending:** indica se o módulo pode ser utilizado em transmissões *broadcast*.
- **discard_on_failure:** indica se o módulo deve descartar a mensagem silenciosamente se a verificação aplicada pelo módulo falhar.

A Figura 27 demonstra como módulos podem ser combinados para satisfazer requisitos de segurança. A *tag* <requirements> indica a seção de configuração dos requisitos e dentro da mesma os requisitos são listados como atributo *name* da *tag* <requirement>. Dentro da *tag* <option> lista-se o conjunto de módulos necessários para assegurar o requisito de segurança criado. No exemplo da Figura 27 o requisito “Confidentiality” é satisfeito usando o módulo “PgpEncryption” e o requerimento “Confidencial_Auditing” é satisfeito usando os módulos “PgpEncryption” e “Log”.

```

<requirements>
  <requirement name='Confidentiality'>
    <option>
      <option_module name='PgpEncryption' />
    </option>
  </requirement>

  <requeriment name='Confidencial_Auditing'>
    <option>
      <option_module name='PgpEncryption' />
      <option_module name='Log' />
    </option>
  </requirement>
</requeriments>

```

Figura 27: Configurando Requerimentos

4.4 Configuração dos Perfis

Um arquivo XML também é utilizado para descrever a composição dos perfis definidos pelo usuário e a ordem de aplicação dos módulos para envio e recebimento de mensagens. No mesmo arquivo, ficam as informações referentes aos nós conhecidos sendo que essa informação pode ser inserida de antemão ou populada através da troca de requisitos. A Figura 28 ilustra a seção do arquivo XML utilizada para descrever um nó conhecido. A seção <peers> contém os vários nós conhecidos e cada nó é identificado pela tag <peer> tendo um nome associado. A tag <available_modules> identifica as informações sobre os módulos disponíveis no nó assim como parâmetros de configuração dos módulos. A tag <required_modules> indica qual módulo precisa ser aplicado para cumprir com as exigências de requisitos do nó remoto.

A Figura 29 ilustra a configuração de perfis. A tag <profiles> envolve os perfis, sendo que estes são listados através da tag <profile>. Dentro de cada perfil, a tag <incoming_requirements> especifica os requisitos de segurança utilizados para o recebimento de mensagens nessa perfil. Em conjunto com essa tag, <incoming_modules> configura os módulos que estão sendo utilizados para cada perfil para aplicação nas mensagens de entrada. Temos a mesma semântica para as tags <outgoing_requirements> e <outgoing_modules>, porém essas indicam módulos aplicados nas mensagens de saída. Por fim, a tag <peers_from> indica os nós que fazem parte desse perfil.

```

<peers>
  <peer name="OPEER1">
    <available_modules>
      <module name="PgpEncryption">
        <parameters>
          <parameter default="739d7f591fc40a55" name="my_encryption_key_id"/>
        </parameters>
      </module>
      <module name="PgpSignature">
        <parameters>
          <parameter default="7199103DADF135EC" name="my_signing_key_id"/>
        </parameters>
      </module>
      <module name="Log"/>
      <module name="Authorization.AuthorizationModule"/>
    </available_modules>
    <required_modules>
      <module name="PgpSignature">
        <parameters>
          <parameter default="7199103DADF135EC" name="my_signing_key_id"/>
        </parameters>
      </module>
    </required_modules>
  </peer>
  <peer name="OPEER2">
    ...
  </peer>
</peers>

```

Figura 28: Informações sobre Nós

```

<profiles>
  <profile name="ProfileA" respect_remote_requirements="true">
    <incoming_requirements>
      <requirement name="Auditory"/>
    </incoming_requirements>
    <outgoing_requirements>
      <requirement name="Authentication"/>
    </outgoing_requirements>

    <incoming_modules>
      <module name="Log">
        <parameters>
          <parameter default="medium" name="log_level"/>
        </parameters>
      </module>
    </incoming_modules>

    <outgoing_modules>
      <module name="PgpSignature">
        <parameters>
          <parameter default="7199103DADF135EC" name="my_signing_key_id"/>
          <parameter default="~/gnupg/pubring.gpg" name="public_keys_file"/>
          <parameter default="~/gnupg/secring.gpg" name="secret_keys_file"/>
          <parameter default="--drytPijTeb3" name="pass_phrase"/>
        </parameters>
      </module>
    </outgoing_modules>

    <peers_from>
      <peer_from name="OPEER5"/>
      <peer_from name="OPEER4"/>
    </peers_from>
  </profile>
</profiles>

```

Figura 29: Configuração de Profiles

4.5 Troca de Requisitos

Devido à natureza dinâmica e assimétrica de redes P2P, P2PSL provê uma forma de alterar as informações de classificação de cada nó da rede. Isso é realizado em três momentos distintos fazendo com que haja pouco esforço por parte do usuário [7].

O primeiro momento acontece quando a configuração inicial da camada é criada. Nesse momento já é possível criar os perfis, configurar os módulos e gerar as chaves necessárias para a utilização com os módulos de criptografia e assinatura. Juntamente com isso, a chave pública é enviada para a ACP (Autoridade de Chave Pública) para outros nós conseguirem acessar a chave pública de maneira segura.

O segundo momento acontece quando o nó ingressa na rede P2P. A P2PSL precisa obter as informações sobre os requisitos de segurança dos outros nós que participam da rede e para isso utiliza um protocolo de negociação de requisitos que será descrito a seguir tendo como exemplo a Figura 30.

Nessa rede, existem três nós utilizando a camada P2PSL. O momento mostrado na figura refere-se à entrada do nó Peer1 na rede composta por Peer2 e Peer3. Peer1 conhece previamente o nó Peer2 por algum contato anterior, mas desconhece Peer3. O protocolo executa os seguintes passos:

1. Peer1 envia uma mensagem *Requeriments_request* para todos os nós da rede pedindo pelos requisitos dos nós presentes na rede;
2. Como Peer2 conhece previamente o Peer1, ele apenas confere a assinatura da mensagem enviada e, se está for correta, envia uma mensagem *Requeriments_request_and_reply* de volta para Peer1 informando seus requisitos com o Peer1 e perguntando quais são os requisitos de Peer1 para com Peer2;
3. Peer3, desconhecendo Peer1, solicita para a Autoridade de Chave Pública (ACP) a chave pública do Peer1. A assinatura da mensagem enviada por Peer1 é conferida usando a chave enviada pela ACP. Se a assinatura estiver correta, Peer3 envia uma mensagem *Requeriments_request_and_reply* para Peer1;
4. Como Peer1 tem conhecimento do Peer2, Peer1 envia a mensagem de *Requeriments_reply* assim que receber *Requeriments_request_and_replay* indicando seus requisitos;
5. Quando Peer1 recebe a mensagem de Peer3, ele irá consultar com a ACP a chave pública de Peer3 para conferir a assinatura da mensagem. Caso a mensagem esteja correta, Peer1 envia para Peer3 os seus requisitos na mensagem *Requeriments_reply*.

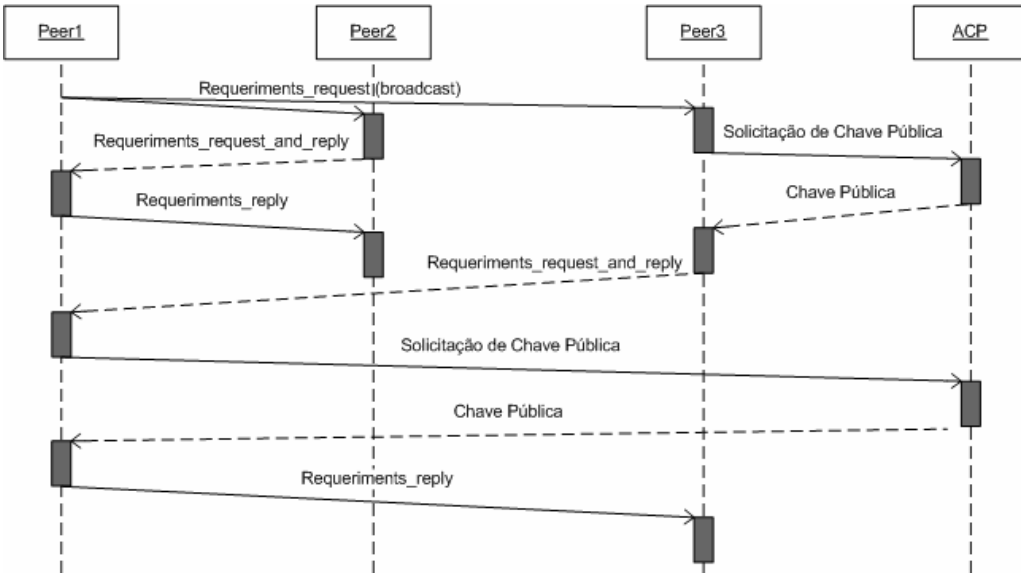


Figura 30: Troca de Requisitos

Depois que os nós trocaram informações pertinentes as suas exigências com outros nós, a configuração de requisitos somente irá ser modificada em um terceiro momento. Isso ocorre quando o usuário da aplicação que utiliza a P2PSL decide modificar os requisitos quanto à comunicação com os outros nós. Essa modificação deve ser repassada para os nós que entram em contato com o nó que fez a modificação nas configurações.

4.6 Implementação

SecurePeer.java é a principal classe a ser utilizada ao se incorporar a P2PSL em uma aplicação P2P. Esta classe contém os métodos necessários para comunicação com os outros nós de forma segura e para inicialização da rede através da JAL/JXTA. Como essa classe é uma extensão da *EZMinimalPeer.java* provida pela JAL para acessar a JXTA de forma abstrata, a classe *SecurePeer.java* deve ser utilizada para todas as tarefas de comunicação e manutenção do substrato. Os principais métodos que devem ser utilizados são:

Métodos de Manutenção do *Overlay*

- *boot(String)*: Método utilizado para iniciar o nó na rede. O parâmetro de entrada é o nome que o nó terá na rede;
- *displayPeers()*: Mostra os nós que estão presentes no grupo;
- *displayGroups()*: Mostra os grupos criados;
- *createGroup(String)*: Cria um novo grupo;
- *joinGroup(String)*: Entra em um grupo já criado.

Métodos para Comunicação com outros Nós

- *sendMessage(String, Message)*: Envia uma mensagem para o nó especificado no primeiro parâmetro;
- *receiveMessage()*: Recebe uma mensagem.

Módulos Implementados

Quando os métodos para comunicação com outros nós são utilizados, a camada automaticamente usa os módulos configurados no arquivo XML de perfis para modificar as mensagens e transmiti-las aos nós destinos. Os módulos que podem ser utilizados são os seguintes:

- **Assinatura PGP:** módulo utilizado para assegurar autenticidade e integridade de mensagens utilizando criptografia assimétrica PGP provida pela biblioteca *BouncyCastle* no *hash* (SHA-1) da mensagem;
- **Criptografia PGP:** esse módulo é responsável por garantir confidencialidade nas mensagens entre usuários e utiliza criptografia simétrica *blowfish* para cifrar a mensagem. A chave randômica gerada para o *blowfish* é cifrada com a chave pública PGP;
- **Verificação de políticas de acesso:** responsável por controle de acesso a recursos. Módulo baseado em *Role-Based Access Control* (RBAC);
- **Geração de logs:** Usado para gerar log das ações e mensagens trocadas. Pode ser usado para auditoria.

5 Análise Crítica da P2PSL

Baseado nos fundamentos literários apresentados até o presente momento, este capítulo fará uma análise sobre aspectos relevantes da P2PSL identificando problemas e limitações. Primeiramente será feita uma análise quanto à arquitetura atual. Posteriormente, a escalabilidade da proposta da camada de segurança será analisada mostrando as limitações para redes com muitos nós. Por fim, será feita uma análise dos ataques conhecidos na literatura e como estes podem afetar a P2PSL.

5.1 Arquitetura

Nesta seção serão discutidas questões arquiteturais e como estas afetam o comportamento do sistema utilizando a P2PSL. As questões avaliadas serão a flexibilidade para comunicação, a proteção que a P2PSL insere e que parte da comunicação fica vulnerável, o controle de composição da rede e a geração de identificadores, utilização com sistema de roteamento em nível de aplicação e a necessidade de utilizar JXTA como substrato.

A Figura 31 ilustra o funcionamento e o fluxo de execução da arquitetura. O desenvolvedor irá atuar somente na camada de aplicação que é referenciada na figura como “JXTA Application”. O código desenvolvido irá se comunicar com a P2PSL apenas através da interface provida pelo *SecurePeer*. É importante frisar que a classe *SecurePeer* é uma extensão da classe *EZMinimalPeer*, que é uma classe da camada de abstração JAL, utilizada para simplificar o emprego da JXTA.

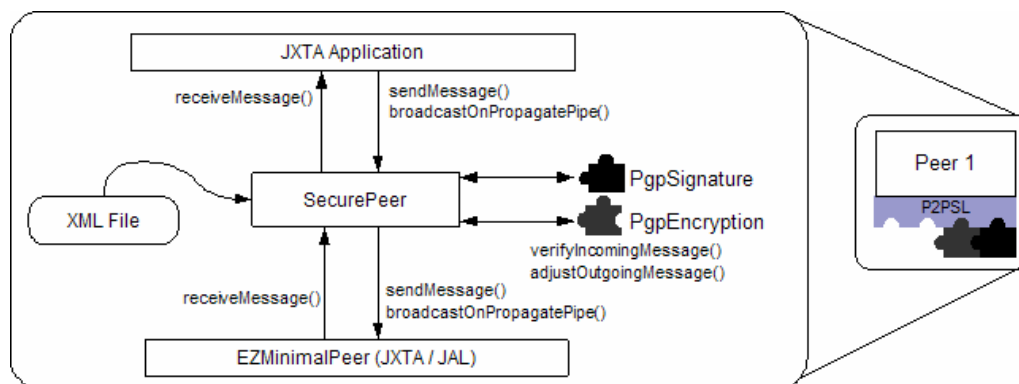


Figura 31: Arquitetura Interna

Devido o utilização de JAL/JXTA, a P2PSL obriga o desenvolvedor a utilizar essa arquitetura e seus protocolos diminuindo a possibilidade de extensão. Atualmente, esse fato ocorre porque o canal de contato com a camada acontece através do *SecurePeer* e este contacta a JXTA diretamente para enviar uma mensagem, ou seja, não é possível utilizar outros métodos de comunicação (que não JXTA) com outros nós da rede sem modificar a implementação da camada. Essa limitação faz com que, em um primeiro momento, somente aplicações que façam uso da JXTA tenham a possibilidade de utilizar a P2PSL para adicionar segurança de maneira modular e gradual.

Em contrapartida, com essa arquitetura é simples criar aplicações exemplo e de baixa complexidade, pois a alta abstração oferecida pelo JAL facilita muitos dos mecanismos de comunicação. Outra vantagem da abordagem atual é a facilidade de implantação e modificação de um sistema que já utiliza JXTA como substrato de comunicação.

Utilizando uma implementação em que o substrato de rede já está construído e implementado faz com que a aplicação fique amarrada ao tipo de comunicação imposta pelo substrato, que neste caso utiliza a JXTA dentro da P2PSL. A implementação atual dificulta a criação de aplicações que tenham controle de composição da rede, ou seja, que necessita controlar em que ponto do grafo de conexão um nó que está entrando deve se conectar. Através dos métodos oferecidos pelo *SecurePeer*, cada nó (aplicação) fica responsável pela criação do seu identificador no substrato, mas o gerenciamento de conflitos fica a cargo da JXTA. Caso um nó resolva adotar um nome presente na rede ou tente burlar a criação de IDs imposta por alguma aplicação na rede, a JXTA será a controladora, ou seja, as restrições impostas no nível de aplicação podem não fazer efeito.

A utilização da P2PSL em cima do substrato de comunicação faz com que não se tenha requisitos de segurança aplicados sobre elementos de controle do substrato como nome do nó destino, nome do nó origem, informações sobre roteamento e outras informações de controle, mas em contrapartida faz com que os requisitos se apliquem sobre os dados da aplicação garantindo requisitos impostos a esses dados.

Com a utilização de JAL/JXTA como substrato (utilizado implicitamente quando os métodos de envio da *SecurePeer* são utilizados), uma aplicação que deseje utilizar roteamento de mensagens pelo *overlay* necessita fazer isso via aplicação, pois JAL não oferece métodos explícitos para tratar dessa funcionalidade. Com isso, os nós de um caminho de roteamento teriam que ler o conteúdo das mensagens para saber qual seria o próximo nó no caminho entre um nó A e um nó B. Isso quebraria o sistema de confidencialidade e aparentemente teria que ser adotada uma outra solução de empacotar duas vezes os dados (como feito em IP sobre IP) para garantir requisitos como o de confidencialidade.

Além disso, existe a necessidade de lidar com algumas funções da JXTA como fazer a inicialização do nó na rede e a criação de grupos de comunicação. Muitas vezes, não é do interesse do desenvolvedor ter envolvimento com essa complexidade de manipulação sendo que a necessidade de fazer contato direto com a JXTA faz com que haja a necessidade de estudar e entender sua API e seu comportamento. Se uma aplicação simples fosse construída, seria necessário conhecimento sobre o JAL/JXTA e seu comportamento, pois falhas e *bugs* inerentes aos métodos de comunicação da JXTA na rede seriam complicados de serem contornados e necessitariam experiência de uso com esse substrato.

5.2 Escalabilidade

A referência [7] demonstra o modelo de segurança utilizado e, aparentemente, o mesmo não foi totalmente explorado e explicitado. Como não foram citados quais seriam as restrições do sistema para registro de chaves, pode-se considerar que o modelo de troca de chaves citado no artigo se enquadra tanto como diretório público quanto autoridade de chave pública citados no Capítulo 3 . Como, nominalmente, citou-se que a entidade seria uma AC, é coerente afirmar que o modelo escolhido é a autoridade de chave pública pelo modo que as mensagens são trocadas entre nós e AC.

Como citado anteriormente, esse modelo centralizador oferece algumas desvantagens como a baixa escalabilidade quando ocorre uma grande quantidade de requisições de chaves para a autoridade. Como o propósito da P2PSL é oferecer uma camada que possa ser utilizada para implantação gradual de requisitos de segurança em ambientes P2P, deve-se ter em mente que uma das principais vantagens de sistemas P2P é a alta escalabilidade do modelo. Dessa maneira, o modelo citado em [7] não é o mais adequado para uma rede com milhões de usuários que demande grande quantidade de consulta de chaves.

Outro problema refere-se a segurança da autoridade. Como ela detém as chaves públicas de entidades do sistema em seu banco de dados, ela pode sofrer ataques de modificações das chaves no banco. As mesmas modificações nas chaves públicas podem ser feitas por funcionários da própria autoridade (por suborno ou sabotagem).

Outro ponto que não foi explorado em [7] é a restrição para o início do sistema, mas supõe-se que as entidades que desejam se comunicar com a autoridade possuam sua chave pública, pois não possuir a chave pública da autoridade deixaria uma brecha para ataques de *man-in-the-middle* ou ainda outras entidades poderiam tentar forjar a autoridade de chave pública.

O protocolo de troca de requisitos descrito em [7] também limita a escalabilidade do sistema quando temos uma rede com grande quantidade de nós. Isso acontece, pois como fora citado, quando um novo nó entra na rede P2P ele deve solicitar os requisitos de segurança dos outros nós da rede em relação a ele. Para realizar esse procedimento, uma mensagem de *broadcast* é enviada para a rede sendo respondida por todos os nós presentes. Dessa forma, o link de saída do nó que está entrando na rede se torna um gargalo para comunicação, pois será necessário enviar uma mensagem para cada nó presente na rede.

Posteriormente, todos os nós que receberam o pedido de requisitos do nó que entrou na rede irão responder ao nó novo fazendo com que seja criado um gargalo no seu link de entrada. Além disso, o novo nó terá que utilizar alto nível de processamento para analisar todos os requisitos enviados pelos nós da rede e fazer a gravação dessas informações em arquivo. Por fim, os nós que enviaram os requisitos para o novo nó e não tinham previamente configurados os requisitos para com este, deverão requisitar (junto com a mensagem de resposta de requisitos) as exigências desse nó. Novamente, o novo nó terá um gargalo no link de saída para responder a todos esses pedidos e alto custo computacional para atender a todas as requisições.

Se a rede P2P apresentar alta transiência de nós em um ambiente com uma quantidade de nós substancial, o protocolo de troca de requisitos irá esgotar a capacidade de comunicação e processamento dos nós da rede, pois cada nó que entra na rede irá gerar, no mínimo, $3N$ mensagens, onde N é o número de nós e o 3 são as mensagens *Requeriments_request*, *Requeriments_request_and_reply* e *Reply*. Deve-se somar a esse número as requisições de chaves públicas feitas à autoridade.

5.3 Segurança

Nesta Seção serão discutidos os tipos de ataques conhecidos na literatura e como estes podem impactar sobre a P2PSL e as aplicações desenvolvidas com sua utilização. Mais especificamente, os principais tipos de ataques expostos serão os apresentados em [34]. É importante salientar que a P2PSL foi pensada como uma camada para incorporação de segurança de maneira a deixar as aplicações *Peer-to-Peer* livres para construírem sua solução final. Essa liberdade permite que vários tipos de aplicações sejam construídas e

o desenvolvedor da aplicação deverá inserir as restrições necessárias ao sistema satisfazendo suas necessidades. A P2PSL auxiliará a criação fornecendo os requisitos de segurança providos por seus módulos.

5.3.1 Negação de Serviço

A negação de serviço afeta a disponibilidade do sistema e pode ser executada de várias formas sendo que cada forma terá uma eficácia e um custo diferente para o atacante. A maneira tradicional de efetuar um ataque desta natureza é o esgotamento das capacidades de comunicação de um nó, ou seja, um nó atacante ou conjunto de nós atacantes enviam uma grande quantidade de dados para o nó atacado. Se atacantes tiverem a sua disposição uma grande quantidade de recursos, será praticamente impossível evitar esse tipo de ataque. Do ponto de vista da P2PSL não há nada que se possa fazer para contornar esse problema, pois o mesmo é um ataque do nível de rede.

Além dos ataques de negação de serviço do nível de rede, existem os ataques que atuam no nível de aplicação onde a intenção é esgotar as capacidades de processamento de uma máquina ou sistema para que não haja respostas a requisições legítimas do sistema. Algumas vezes, os ataques que atuam nesse nível tentam fazer com que a aplicação de um nó atacado se comporte arbitrariamente e produza dados inválidos na rede introduzindo alta sobrecarga. A P2PSL não oferece proteção direta contra esse tipo de ataque, pois ela somente suporta a garantia da origem da mensagem e um nó conhecido pode fazer esse tipo de ataque.

A P2PSL não oferece suporte para situações onde nós respondem corretamente as buscas de informações, mas se negam a entregar o conteúdo solicitado, pois não existe módulos que façam controle de reputação. Em contrapartida, o módulo de autenticação provido pela P2PSL evita ataques de injeção de mensagens não solicitadas na rede, pois mensagens não poderão ser forjadas e então serão ignoradas pelos outros nós que estejam utilizando a P2PSL.

Outro ataque descrito é o ataque de nó lento (*slow node attack*). Esse ataque consiste em um nó malicioso interceptar mensagens de nós não-maliciosos modificando-as de forma a falsificar informações sobre capacidade de banda e processamento. A falsificação faz com que os receptores destas mensagens pensem que um nó tem uma capacidade maior do que a real, com isso o sistema pode tentar usar esse nó para desafogar certa parte da rede causando um prejuízo tanto ao nó quanto aos interessados em seus serviços. O módulo de autenticidade resolve facilmente esse problema, pois as mensagens, depois de assinadas, não podem ser modificadas.

5.3.2 Roteamento

Esse tipo de ataque afeta o substrato de rede das aplicações P2P e tem o intuito de fazer com que o sistema fique lento ou não responda as requisições. Esse ataque acontece quando nós maliciosos “envenenam” as tabelas de roteamento de nós corretos ou quando nós maliciosos rateiam as mensagens por caminhos errados ou para nós inválidos, fazendo com que as mensagens se percam na rede.

Como a P2PSL foi proposta para atuar na camada de aplicação, a maioria dos ataques dessa natureza não poderá ser impedido ou detectado por ela. Dessa forma, outros métodos precisam ser desenvolvidos para contornar os problemas gerados por esse tipo de ataque, tanto em nível de aplicação quanto em nível de rede.

A implementação atual da P2PSL está baseada no substrato de rede JXTA e, portanto os ataques que podem comprometer o JXTA também serão capazes de comprometer a

P2PSL. Porém, pode-se incluir mecanismos de controle no nível da aplicação na tentativa de barrar alguns dos ataques. A seguir, os ataques de roteamento serão descritos analisando o impacto para a P2PSL.

[35] descreve três tipos de ataques desta natureza em redes DHT. O primeiro acontece quando um nó atacante que faz parte do *overlay*, ao receber mensagens, encaminha as mesmas para nós inexistentes na rede ou as envia por rotas incorretas. Com isso, as mensagens se perdem pela rede, mas os nós não irão desconfiar do nó atacante, pois ele está apenas encaminhando as mensagens. O segundo tipo acontece quando nós maliciosos inserem mensagens de atualização de rotas no *overlay* de forma a enganar os nós corretos na rede sobre a rota de outros nós. Com isso, nós sadios irão enviar mensagens por caminhos incorretos, pois foram enganados e tiveram suas tabelas de roteamento poluídas.

O terceiro trata de partições na rede que ocorrem quando nós maliciosos formam uma rede paralela e fazem com que, erroneamente, um nó sadio entre nessa rede. Com o controle da rede, os nós maliciosos podem enganar nós sadios negando-lhes serviços ou ainda distribuindo conteúdo poluído. Outra consequência desse ataque, também conhecido como Eclipse, é que a rede paralela pode conter nós que fazem a ligação entre a rede falsa e a rede normal monitorando os nós sadios para verificar que tipo de buscas eles estão fazendo. Isso termina com a anonimidade de uma rede.

Para efetuar um ataque do tipo Eclipse pode-se utilizar duas abordagens. A primeira diz respeito à manipulação do algoritmo de manutenção do *overlay* onde nós maliciosos irão envenenar tabelas de roteamento ou tabelas de nós disponíveis para conexão enganando parte da rede. A segunda forma diz respeito à utilização de ataque Sybil [36] para conseguir grande quantidade de identificadores para controlar uma porção da rede.

P2PSL não fornece suporte direto para resolver esses problemas, pois os mesmos atuam diretamente no substrato de rede. Sybil é um problema de alocação de identificadores e isto não é coberto pela P2PSL, assim como roteamento. Porém, com a utilização do módulo de autenticidade e a possibilidade de descobrir se alguma mensagem está envenenada é possível determinar quem está gerando essas mensagens e colocá-lo em uma lista negra.

5.3.3 Confidencialidade

A confidencialidade tem por objetivo não permitir que um nó qualquer da rede consiga identificar e interpretar a comunicação sendo realizada por outros dois nós da rede. Isso impede que se descubra a informação que é trocada entre dois nós como mensagens pessoais ou mesmo arquivos. Garantir essa propriedade utilizando a P2PSL é simples, pois basta utilizar o módulo de confidencialidade informando para o mesmo a chave pública do nó que se quer contactar, configurar os perfis de acordo para a utilização do módulo com o nó desejado e a P2PSL irá criptografar as mensagens de comunicação com o nó escolhido garantindo a confidencialidade.

5.3.4 Autenticidade

Assegurar identidade de um nó na rede é uma tarefa difícil de ser resolvida definitivamente, pois sempre haverá a necessidade de confiarmos em alguém ou alguma entidade e neste momento a vulnerabilidade aparece. Um nó pode assegurar digitalmente a autenticidade de outro nó, basicamente, de três maneiras distintas: agência de confiança, próprio nó ou outros nós.

Teoricamente, é fácil assegurar que uma mensagem não é forjada ou garantir o remetente de uma mensagem, pois para isso basta utilizar assinaturas digitais. O problema se encontra em como verificar a quem essa “identidade digital” pertence. Fazendo uso deste problema, o ataque Sybil acontece quando um único nó pode criar várias identidades digitais dentro da rede P2P para enganar outro nó.

Apesar da P2PSL garantir a autenticidade dos remetentes das mensagens, ela não é capaz de detectar se um ataque do tipo Sybil está presente na rede. Isso acontece, pois um nó pode gerar vários pares de chave pública/privada e depositar essas chaves na autoridade de chave pública. Depois disso, o nó pode instanciar várias vezes a aplicação P2P localmente e utilizar para cada uma delas um dos pares de chaves criados. Esse nó de posse das várias identidades pode comprometer algoritmos de replicação, pois várias das réplicas podem acabar sendo alocadas na mesma entidade física que possui várias entidades digitais.

A P2PSL também não oferece suporte para impedir que nós gerem suas identidades. Essa restrição fica a cargo da aplicação, mas vale ressaltar que é difícil impedir que um nó instancie várias vezes a aplicação para conseguir várias identidades.

5.3.5 Reputação e Confiança

Aparentemente, não existe, hoje, um sistema de reputação que resolva os vários problemas provenientes de nós que não querem cooperar (agindo de má fé) ou de nós que estão atacando a rede ativamente. Existem na literatura muitas propostas tentando endereçar alguns dos problemas de reputação, porém eles apresentam fraquezas e não resolvem o problema como um todo.

O problema de reputação não acontece apenas no mundo digital, mas também no mundo real onde pessoas podem enganar outras, agir de má fé, passar informações errôneas ou caluniosas. O grande dilema está em quem confiar quando se trata de uma interação com outro nó. A única informação realmente segura que se tem é quando existe conhecimento e interação prévia com o nó, pois não podemos garantir que outros nós enviarão informações confiáveis sobre o nó em questão. Mas e quando se interage com um nó pela primeira vez? Qual a abordagem deve ser tomada?

Quando se pensa nesses sistemas é necessário avaliar qual *tradeoff* entre a imposição de mecanismos para controlar todas as variáveis em um sistema de reputação e a utilização do sistema, pois a utilização pode ficar bastante deteriorada dependendo da quantidade de restrições feitas.

Existem muitos casos em que sistemas de reputação são interessantes para garantir alguns requisitos ou a melhor utilização do sistema como um todo. Um exemplo disso é utilizar reputação para beneficiar ou punir um usuário dependendo da sua colaboração com a rede. Assim um nó que tentasse atuar como um *free-rider* apenas consumindo recursos começaria a ser barrado com o tempo até que contribuísse com os outros nós.

Outro exemplo é o de comércio eletrônico como o eBay e o Mercado Livre onde os usuários do sistema classificam uns aos outros. Essa classificação ocorre a partir da experiência que um vendedor teve com um comprador. Geralmente, somente usuários com boa qualificação conseguirão vender seus produtos sem problemas ou desconfiança. Nesse modelo, o eBay ou Mercado Livre atuam como gerentes de reputação dando certa garantia aos outros nós sobre as informações que ele distribui. Mesmo com esse tipo de sistema, ainda existe o problema de traição, onde um nó tem comportamento correto durante um período de tempo e, quando ganha bastante confiança, aplica um golpe prejudicando outro usuário.

Além da traição, existe outro ataque que tem uma eficácia elevada. Este ataque utiliza o conluio entre nós para prejudicar um usuário. Como exemplo, pode-se citar um grupo de 200 usuários que resolve prejudicar o nó X. Os duzentos usuários escolhem um nodo Y que será o nó em que eles irão assegurar confiança provendo falso testemunho ele dizendo que Y é um nó correto. Quando X fizer uma transação com Y ele irá considerar que Y é um nó correto pela quantidade elevada de recomendações, porém Y é um nó malicioso do grupo e irá prejudicar X.

O ataque de Sybil pode ser usado juntamente com esse citado no parágrafo anterior de forma a não existir a necessidade de vários nós entrarem em conluio. Para isso, um único nó aplicando Sybil seria capaz de criar todas as recomendações falsas e elevar falsamente a confiabilidade de um nó. Assim, mesmo utilizando um gerente de reputação, é complicado o gerente garantir quais mensagens são verdadeiras e quais são forjadas (conluio).

Ao contrário dos ataques citados anteriormente, que aumentam falsamente a reputação de um nó, um ataque pode visar prejudicar a reputação de um nó correto no sistema para que este fique incomunicável ou seja ignorado. Esse ataque pode ser realizado tanto por um conluio quanto por um atacante usando Sybil. O ataque consiste em informar, falsamente, para outros nós que um nó em questão (atacado) não é confiável anulando a confiabilidade dele perante os outros nós.

Como pode ser visto, existe um grande desafio a ser vencido na área de reputação de entidades. As várias propostas da literatura resolvem apenas alguns pontos isolados dos problemas ainda deixando brechas para atacantes. A P2PSL não fornece mecanismos explícitos para lidar com reputação, logo aplicações desenvolvidas utilizando-a poderão sofrer os ataques descritos aqui a menos que implementem mecanismos próprios para controlar esses requisitos.

5.3.6 Autorização

Quando se deseja criar sistemas que são mais que simples redes de compartilhamento de arquivos sem nenhum controle é necessário empregarmos mecanismos que permitam estabelecer controle de acesso aos recursos [34]. Sistemas de controle de acesso provém uma forma de controlar quais entidades da rede terão acesso a quais recursos e em qual momento. Assim, recursos não ficarão disponíveis a qualquer entidade presente na rede.

A natureza autônoma das redes P2P fazem com que sistemas que usem listas globais de controle de acesso centralizadas em servidores não sejam factíveis, pois cada nó precisa informar quais são as suas restrições para acessar os recursos que ele disponibiliza [37]. Além de listas de controle individuais, os nós precisam de políticas diferentes das tradicionais baseadas em sistemas fortemente acoplados, pois a comunicação em sistemas P2P baseia-se em uma interação com muitos nós não conhecidos ou anônimos.

Outro ponto importante é encorajar os participantes da rede a utilizá-la, pois políticas de controle de acesso diminuirão a disponibilidade de recursos ou arquivos na rede de maneira a reduzir a possibilidade de encontrar o que se deseja. É importante que o controle de acesso enderece esse problema de maneira a incentivar os participantes a compartilharem recursos provendo um bônus que ele possa utilizar no futuro.

A P2PSL possui um módulo baseado em RBAC para políticas de controle de acesso. Na política se define quais nós desempenham quais papéis na rede e cada papel terá associado os recursos que serão liberados e bloqueados. A política de acesso atual é baseada no nome dos nós, por isso pode ser atacada se um nó obtiver um dos nomes contido na lista de acesso. Na P2PSL é simples obter um nome na rede, pois é a própria

implementação que defini o nome. Dessa forma, um nó pode acessar qualquer recurso de outro nó se souber o nome de nó que tenha esse recurso liberado. O módulo RBAC não trabalha em conjunto com módulos de autenticidade para evitar isso.

5.3.7 Integridade

As aplicações mais comuns no ambiente P2P são as de compartilhamento de arquivos e por esse motivo é importante que exista um meio de garantir a integridade dos arquivos de maneira que usuários não obtenham conteúdo corrompido. A adulteração do conteúdo pode ocorrer tanto no seu armazenamento quanto no trânsito do mesmo pela rede.

Para assegurar que um conteúdo não seja modificado no trânsito pela rede basta utilizar o módulo de autenticidade, pois caso o conteúdo seja modificado a assinatura não irá corresponder ao conteúdo e isso invalidará o dado. A garantia ocorre, pois o módulo de autenticidade (assinatura digital) aplica o algoritmo de criptografia sobre o *hash* do conteúdo. Dessa forma, se o conteúdo for alterado, seu *hash* não será o mesmo afetando o valor da assinatura.

Em muitos sistemas de compartilhamento de arquivos, uma requisição é respondida por várias fontes aumentando a disponibilidade, dessa maneira o nó requisitor irá contactar um subgrupo dos nós que detém o conteúdo que ele deseja. Caso haja algum nó malicioso no caminho entre o nó requisitor e algum dos nós entregando o conteúdo alterando o conteúdo, o nó requisitor pode solicitar o mesmo conteúdo de outro dos nós que o detém.

5.3.8 Anonimidade e Negabilidade

Segundo [38], a anonimidade de um sistema deve cobrir os seguintes aspectos:

- **Anonimidade do Autor:** Impede um atacante de descobrir quem é o autor de um documento;
- **Anonimidade do Publicador:** Impede um atacante de descobrir quem publicou o documento na rede;
- **Anonimidade do Leitor:** Impede que um atacante consiga identificar quais são os leitores de um documento de maneira a proteger sua privacidade;
- **Anonimidade do Servidor:** Dado um documento, impede que um atacante consiga identificar qual ou quais servidores na rede tem posse deste documento;
- **Anonimidade do Documento:** Esse aspecto também é conhecido como negabilidade, pois impede o servidor de identificar o conteúdo dos documentos que ele armazena;
- **Anonimidade de Consulta:** Impede que um servidor consiga identificar o tipo de consulta que um cliente está fazendo. Da mesma forma, impede que o servidor identifique os dados que está enviando para o cliente em uma resposta.

A anonimidade tem a intenção de manter a privacidade dos usuários da rede e, juntamente com isso, evitar censura e problemas legais devido a conteúdo que possa ser considerado legalmente ofensivo em alguns países. Um exemplo de problema legal foi o caso Napster [39] em que a empresa que centralizava o servidor para permitir aos usuários troca de músicas no formato MP3 foi processada e teve que mudar seu modelo de negócio e controlar o que seus usuários estivessem distribuindo.

Como exemplo de censura pode-se citar a proibição ao acesso e distribuição a certos documentos, vídeos e documentários. Um exemplo disso no Brasil é o documentário “Muito Além do Cidadão Kane” e na China pode-se citar uma grande gama de informação que é considerada proibida sendo que o governo chinês possuiu um acordo com a Cisco para controlar o tráfego de Internet no país.

A P2PSL não oferece suporte para mecanismos de anonimidade. Atualmente é necessário implementar soluções para isso totalmente na camada da aplicação e por esse motivo é possível que um atacante obtenha informações sobre quem está fazendo o que em um sistema utilizando como único mecanismo de segurança a P2PSL.

5.3.9 Poluição de Conteúdo

Esse assunto pode ser considerado juntamente com a integridade de arquivos, pois visa fazer com que o conteúdo chegue corrompido para um usuário. A poluição de conteúdo tem um grande apelo comercial, pois entidades como a indústria fonográfica estão tentando barrar a pirataria de músicas. Existem entidades que oferecem serviços de poluição como o Overpeer (<http://www.overpeer.com>).

Basicamente, os motivos que levam a inserção de conteúdo poluído na rede podem ser divididos em três: usuários maliciosos com intuito apenas de prejudicar a rede; usuários com intenção de espalhar vírus ou material próprio sob títulos de alta popularidade; indústria tentando proteger sua propriedade intelectual da disseminação sem os devidos pagamentos.

Existem certos tipos de poluição que não podem ser detectados pelo sistema, ficando a cargo do usuário a classificação manual do conteúdo como poluído. Dessa forma, para melhor funcionamento da rede, o sistema deve permitir e facilitar a classificação do conteúdo pelos usuários para que possam informar uns aos outros a situação de certo conteúdo ou nó como poluidor. Outra boa prática do ponto de vista do usuário é apagar imediatamente um conteúdo que veio poluído para não disseminá-lo entre os outros usuários e não ser taxado como poluidor.

A P2PSL não oferece meios para controlar a distribuição de conteúdo poluído. Da mesma forma, não é capaz de taxar usuários poluentes como maliciosos e fica a mercê desse tipo de ataque.

6 Proposta de Melhorias

Nesse capítulo serão discutidas propostas de melhorias para P2PSL no que diz respeito aos problemas levantados no capítulo anterior. Primeiramente serão feitas propostas sobre modificações na arquitetura expondo as conseqüências de cada modelo. Depois disso serão propostas alterações para reduzir os problemas de escalabilidade já elucidados. Por fim, serão feitas propostas na tentativa de resolver problemas de vulnerabilidades encontradas.

6.1 Arquitetura

Como a P2PSL está fortemente acoplada a JXTA, por questões de interoperabilidade, é interessante fazer com que a P2PSL possa ser utilizada sem necessariamente utilizar a JXTA como substrato de rede. Para isso, retira-se a JXTA fazendo modificações na classe *SecurePeer.java*. Isso deixaria espaço para implementação da biblioteca da P2PSL de duas formas distintas: intermediando o acesso ao substrato de rede; atuando lado a lado com a aplicação sem contactar o substrato de rede.

Para deixar a possibilidade de escolha para o programador em utilizar a P2PSL com ou sem JXTA, pode-se modificar o comportamento dos métodos da classe *SecurePeer.java* de forma que se o desenvolvedor quisesse utilizar JXTA como substrato utilizaria um método *init()* fazendo as inicializações do substrato e utilizaria o método atual *send()* para enviar as mensagens. Para o caso onde o desenvolvedor não quisesse utilizar JXTA seria disponibilizado um *wrapper* que faria as mesmas modificações na mensagem que o método *send()* faz, porém não enviaria a mensagem devolvendo a mesma para aplicação que decidiria como enviá-la.

Retirando a obrigatoriedade de utilizar JXTA como substrato de comunicação deixaria a P2PSL mais flexível permitindo a incorporação dela em sistemas sem JXTA e permitindo a criação de algoritmos que atendessem aos requisitos impostos pelo desenvolvedor. Um exemplo disso são algoritmos de roteamento que devido à utilização da JXTA são complicados de serem desenvolvidos e integrados. A opção de não utilizar JXTA facilitaria a resolução de problemas caso houvessem problemas de comunicação na JXTA.

Nessa diversidade de modelos existem questões pertinentes que devem ser observadas quando se deseja modificar a camada estruturalmente ou tentar utilizá-la de maneira a melhor obtenção de resultados para uma aplicação específica. As próximas subseções irão apresentar diferentes meios de comunicação que poderiam ser utilizados com a P2PSL. Serão analisadas como as mensagens de cada uma das camadas (aplicação, P2PSL e substrato) seriam afetadas pela decisão de arquitetura. Assim como as implicações referentes à implementação em um sistema tanto Estruturado quanto Não-Estruturado.

6.1.1 P2PSL entre Aplicação e Substrato de Rede

Esse modelo é o mesmo utilizado atualmente pela P2PSL, porém com a eliminação da JXTA e algumas modificações estruturais pode-se contemplar quatro variações do modelo. Todas as variações irão prover os mesmos requisitos e não mudarão o

comportamento da camada do ponto de vista do usuário final, porém as principais mudanças ocorrerão na maneira como o desenvolvedor irá utilizar a P2PSL e como a mesma será implementada.

Todas as menções feitas nesta Subseção serão sobre a Figura 32. A figura está dividida em quatro partes e cada uma dessas partes representa uma das variações. Na figura, as setas vermelhas indicam as mensagens que são de uso exclusivo da P2PSL (troca de requisitos e chaves) e a figura está subdividida da seguinte maneira:

- Representa a utilização da P2PSL exatamente como no modelo atual. Ela intercepta todas as mensagens da camada de aplicação fazendo as modificações necessárias para atingir os requisitos de segurança e posteriormente contactando diretamente o substrato de rede para envio.
- A diferença para o modelo (a) está na possibilidade que a aplicação tem de contactar diretamente o substrato de rede sem passar, necessariamente, pela P2PSL. Neste caso, a P2PSL usaria o substrato apenas para enviar as mensagens de troca de requisitos e a recuperação de chaves. A aplicação contactaria a P2PSL para ajustar a mensagem e a P2PSL devolveria a mensagem ajustada para a aplicação decidir o que fazer com a mesma.
- Este modelo não permite que a P2PSL se comunique diretamente com o substrato de rede, ou seja, toda comunicação que entra ou sai da P2PSL passará pela camada de aplicação.
- Este modelo representa a utilização da P2PSL como uma Arquitetura Orientada a Serviço (*Service Oriented Architecture – SOA*). Neste caso, o protocolo SOAP [40] é utilizado para comunicação entre a camada e outros componentes do sistema.

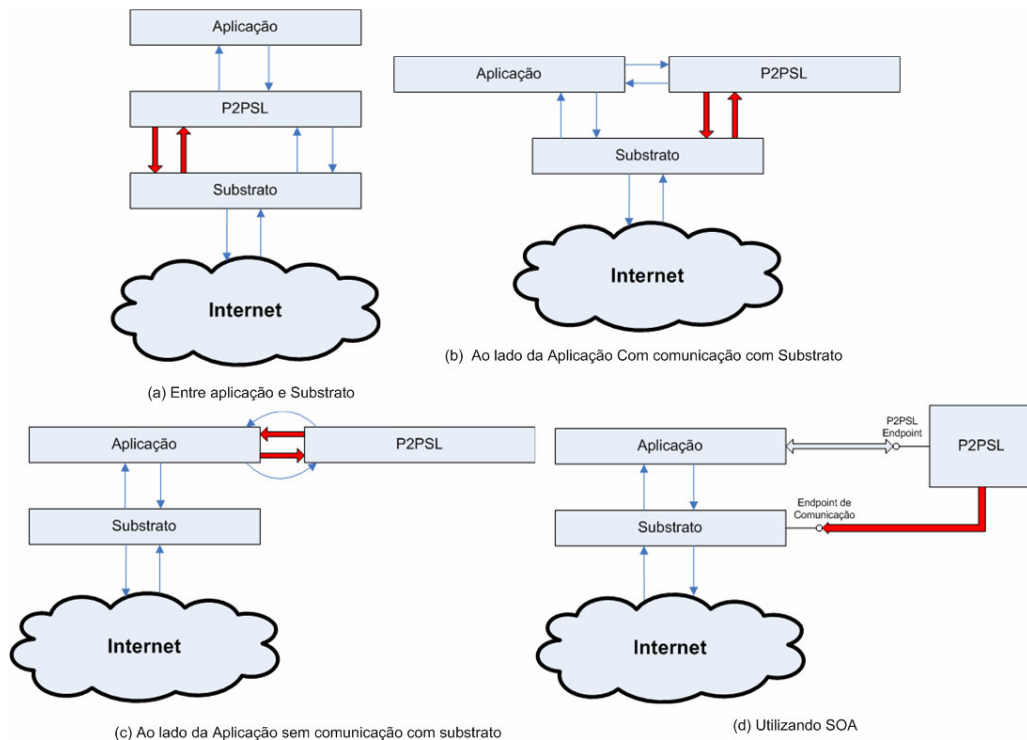


Figura 32: P2PSL atuando entre Aplicação e Substrato de Rede

A utilização dessa arquitetura pode ser feita tanto em redes Estruturadas quanto em redes Não-Estruturadas garantindo os requisitos de segurança providos pela P2PSL. Aparentemente, só não é possível garantir controle de acesso em um ambiente com anonimidade, pois um requisito fere o outro. As limitações citadas no Capítulo 5 sobre utilização em ambiente Estruturado acontecem devido à utilização da JXTA como substrato sem a possibilidade de configurações adicionais de como utilizá-la, porém se um substrato que faça roteamento sem passar pela camada de aplicação for implementando, os requisitos serão respeitados no caminho de roteamento.

Mensagens da P2PSL

As mensagens que são enviadas exclusivamente pela P2PSL são as utilizadas para realizar o protocolo de troca de requisitos e recuperação de chave pública de entidades descritos no Capítulo 4 . Os modelos de arquitetura (a), (b) e (d) tratam o envio dessa mensagem de forma simples, pois contactam o substrato de rede diretamente para que este envie as mensagens para outros nós.

Por outro lado, no modelo (c) existe a necessidade de controle dessas mensagens por parte da aplicação. Nesse caso, a aplicação, quando utilizasse a API da P2PSL, seria notificada caso houvesse a necessidade de trocar requisitos ou chaves. De posse da notificação, a aplicação teria que encaminhar a mensagem para o substrato de rede e entregar a resposta para a P2PSL.

Mensagens da Aplicação

Em todos os casos é simples utilizar a P2PSL para ajustar as mensagens que serão enviadas para outros nós e para isso basta enviá-las para a P2PSL. O modelo (a) é o único que obriga a passagem de todas as mensagens enviadas pela aplicação pela camada de segurança, pois a P2PSL se encontra entre a aplicação e o substrato formando uma pilha.

Os outros modelos são mais flexíveis [(b), (c) e (d)], pois permitem que a aplicação contacte o substrato de rede diretamente quando necessário. Isso não significa que o modelo (a) seja o pior nesse caso, mas que ele restringe mais a aplicação. O desenvolvedor deverá avaliar se prefere um modelo mais restrito ou mais flexível dependendo de suas necessidades.

Mensagens do Substrato

Em nenhum dos casos as mensagens de roteamento e controle do substrato são afetadas. Elas trafegam livremente pela rede sem a interferência da P2PSL, ou seja, as mensagens do substrato não possuirão requisitos de segurança.

Características de Implementação

No modelo (a) e (b), o substrato precisa ser feito estendendo ou implementando classes já definidas pela P2PSL. Isso é necessário, pois no código da P2PSL é preciso ter uma chamada a algum tipo de método *send()* que envie dados através do substrato, ou seja, pelo menos uma classe conhecida da P2PSL o substrato deverá implementar.

No caso (c) e (d), o substrato não necessitará implementar classes conhecidas da P2PSL. Em (c) porque não existe comunicação direta entre camada de segurança e substrato de rede e em (d) porque a comunicação acontece via SOAP. Dessa forma, é importante frisar que a escolha pelo modelo baseado em SOA coloca a necessidade de desenvolvimento de mecanismos de comunicação SOAP tanto da aplicação com a P2PSL quanto da P2PSL com o substrato.

Utilizar SOA torna a camada mais independente de linguagem/ambiente aumentando sua usabilidade, pois qualquer aplicação desenvolvida com suporte a protocolo SOAP poderá utilizar a camada de segurança. Isso elimina a necessidade de “ganchos” para intercomunicar aplicação que não executa com código nativo Java com a P2PSL.

6.1.2 P2PSL embaixo do Substrato de Rede

Esse modelo visa prover os requisitos de segurança não somente nos dados da aplicação, mas também nos dados do substrato de rede. Para isso, utiliza-se a P2PSL abaixo do substrato de rede ou “ao lado dele”, pois isso faz com que os dados pertinentes ao substrato sejam tratados pela camada de segurança. Para resolver isso, pode-se aplicar algumas variações do modelo e, como na arquitetura anterior, terão alguns comportamentos deferentes sobre as mensagens, tipos de redes suportadas, implementação e comportamento.

Todas as menções feitas nesta Subseção serão sobre a Figura 33. A figura está dividida em três partes e cada uma dessas partes representa uma das variações da arquitetura. Na figura, as setas vermelhas indicam as mensagens que são de uso exclusivo da P2PSL (troca de requisitos e chaves) e a figura está subdividida da seguinte maneira:

- a. A utilização da P2PSL abaixo do substrato de rede contactando diretamente a interface de rede faz com que todas as mensagens enviadas pelo substrato tenham que passar pela camada de segurança. Isso inclui uma complexidade a mais na camada segurança, pois ela terá que gerenciar conexões de alguma forma.
- b. Esse modelo, como na Subseção 6.1.1, mostra a P2PSL sendo utilizada como Arquitetura Orientada a Serviço, porém quem contacta a camada de segurança utilizando SOAP é o substrato de rede e não a aplicação.
- c. Como no modelo (b), a camada não irá contactar a rede e apenas será contactada pelo substrato onde o substrato terá a opção de contactar diretamente a rede ou utilizar a P2PSL quando for necessário.

A utilização dessa arquitetura pode ser feita tanto em redes Estruturadas quanto em redes Não-Estruturadas, porém deve-se observar limitações em sua utilização em redes Estruturadas que usam os nós da rede para rotear as mensagens. Utilizar a camada de segurança sobre todo o conteúdo (roteamento e dados) de uma única vez faz com que todos os nós que precisem rotear uma mensagem sejam capazes de ler seu conteúdo. Isso quebra o requisito de confidencialidade entre dois nós A e B, pois todos os nós que estão entre A e B roteando as mensagens serão capazes de ler o conteúdo.

Os requisitos de segurança serão garantidos entre um par de nós vizinhos no caminho de roteamento, porém os requisitos não serão satisfeitos entre os nós das extremidades do caminho. Tome como exemplo o modelo (a) e a aplicação do nó A tentando comunicar-se com a aplicação do nó B tendo um nó C entre eles que irá rotear as mensagens. Quando a aplicação A requisitar para o substrato para enviar os dados, esse irá enviar a mensagem para C. C precisa ler a mensagem para saber o que fazer com a mesma e isso impede a confidencialidade entre A e B. Como a P2PSL localiza-se embaixo da pilha, no caso de utilização do módulo de autenticidade, a autenticidade será garantida apenas entre os nós A-C e C-B, mas não entre os nós A-B.

Em redes Estruturadas com roteamento pode-se utilizar os modelos (b) e (c), mas para isso é necessário que o substrato não utilize requisitos de segurança na informação referente ao nó destino da mensagem para que os nós intermediários de um caminho

possam rotear a mensagem sem descriptografá-la. Para redes Não-Estuturadas, onde a conexão entre nós é feita diretamente entre nós sem a necessidade de nós roteadores, o modelo pode ser utilizado sem essas limitações.

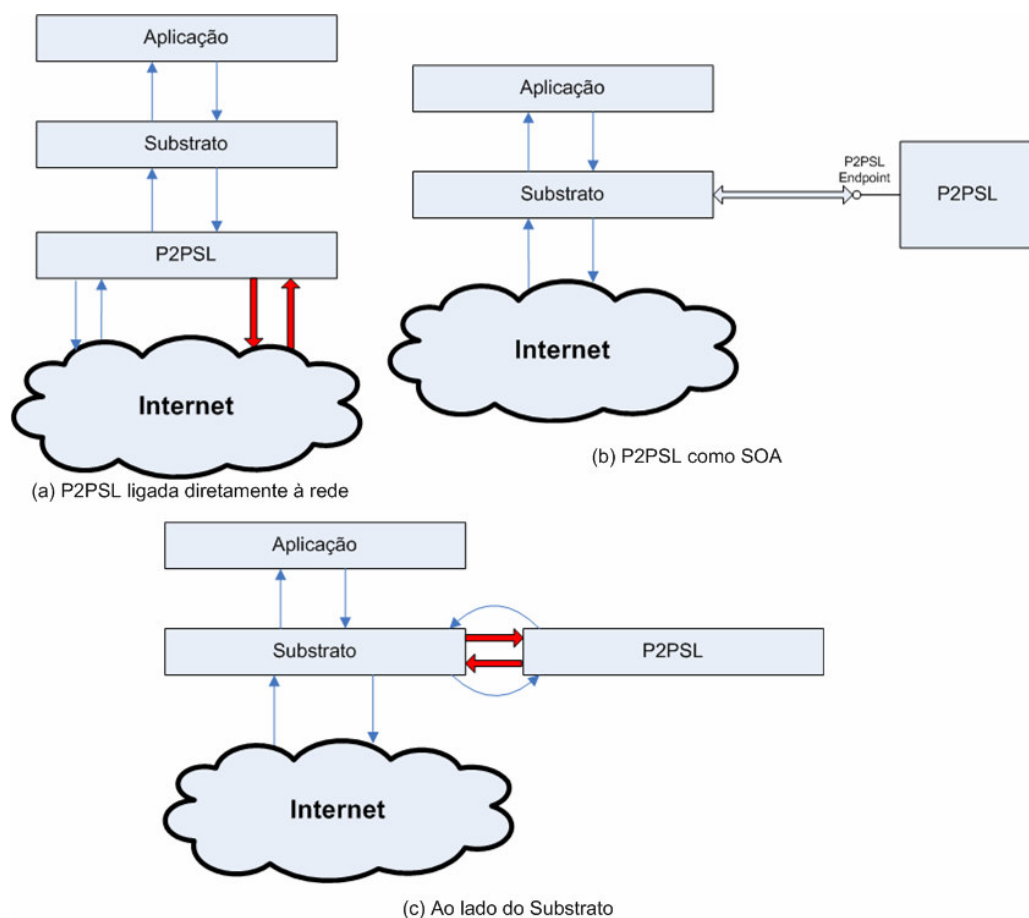


Figura 33: P2PSL atuando abaixo do Substrato de Rede

Mensagens da P2PSL

No modelo (a) as mensagens da P2PSL são enviadas para rede diretamente pela camada sem nenhuma interferência de outra das camadas. Isso faz com que a troca de requisitos e chaves seja feita de maneira transparente ao resto da aplicação, pois nos outros nós a P2PSL irá interceptar essas mensagens e respondê-las sem envolver outras camadas.

Para o modelo (b) e (c) a abordagem tomada deve ser diferente, pois como a P2PSL não terá acesso direto a rede, é necessário que o substrato de rede gerencie as mensagens da camada de segurança. Para o modelo (c) a gerência deve ocorrer através da API Java e para o modelo (b) a gerência das mensagens da P2PSL deve ocorrer através de diferentes mensagens trocadas através do *endpoint* usando SOAP.

Mensagens da Aplicação

O fluxo de mensagens que saem da aplicação, em todos os casos, sempre vai para o substrato de rede, ou seja, não existe muita coisa que pode ser mudada nesse aspecto. As restrições que são necessárias observar são as referentes às limitações da utilização de requisitos de segurança fim a fim citados anteriormente no caso de utilização de algoritmos de roteamento.

Mensagens do Substrato

No modelo (a), as mensagens enviadas pelo substrato de rede sempre serão interceptadas pela P2PSL que irá aplicar os módulos de segurança necessários para cumprir os requisitos configurados no perfil. Mensagens de roteamento e controle do substrato sempre obedecerão aos requisitos de segurança, pois passarão pela P2PSL.

Diferentemente do modelo (a), nos modelos (b) e (c) existe a possibilidade do substrato contactar a rede sem a necessidade de aplicação de módulos de segurança. Isso abre espaço para utilização de mensagens dos algoritmos de manutenção do *overlay* sem a necessidade de passar pela P2PSL, optando por utilizar a P2PSL apenas quando houver requisição de envio de dados a partir da camada de aplicação.

Características de Implementação

Diferentemente da arquitetura citada na Subseção 6.1.1, esta arquitetura não precisa utilizar extensão de classes e interfaces pré-definidas pela camada de segurança, pois nos três casos ilustrados na figura o contato pode ser feito simplesmente instanciando a P2PSL e utilizando seus métodos. No caso do modelo (b) utiliza-se a instanciação de uma classe de comunicação com o *endpoint* da P2PSL.

Como o modelo (b) utiliza a P2PSL como SOA, existe a necessidade de utilizar uma implementação de protocolo SOAP para comunicação com a camada de segurança. E como neste caso a camada não pode acessar a rede, o substrato deve gerenciar as mensagens de troca de requisitos e chaves que a P2PSL necessite enviar. Quando o substrato receber mensagens através da rede que sejam de utilização da P2PSL, é necessário passar essas mensagens diretamente para a camada de segurança para que o protocolo utilizado por ela seja completado.

6.1.3 P2PSL Protegendo Aplicação e Substrato de Rede

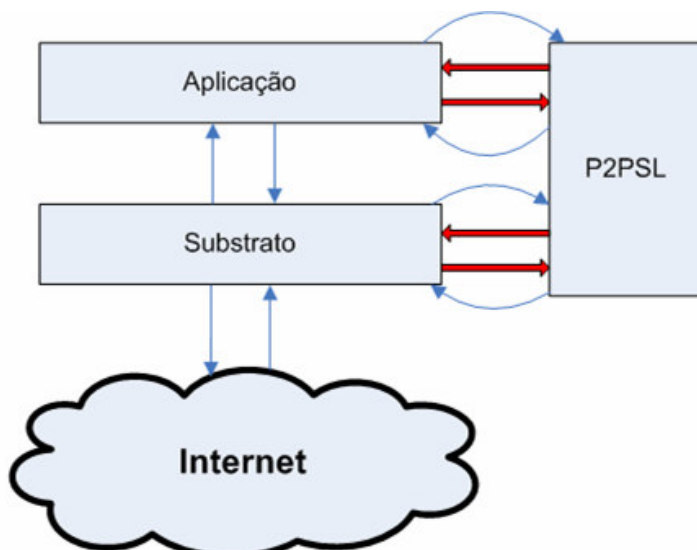
Esse modelo visa prover os requisitos de segurança tanto para os dados da aplicação quanto para os dados do substrato de rede. Mas, diferentemente do modelo anterior, esse modelo aplica os requisitos de segurança de maneira independente entre dados da aplicação e substrato de rede. O objetivo é poder utilizar os requisitos de segurança fim a fim, ou seja, da aplicação origem para aplicação destino, mas juntamente aplicar os requisitos também entre os nós que estão roteando as mensagens na rede, ou seja, entre nós vizinhos. Para isso, utiliza-se a P2PSL em dois momentos distintos: o primeiro deles é aplicar a P2PSL na mensagem da aplicação; o segundo momento é aplicar a P2PSL na mensagem do substrato de rede.

Todas as menções feitas nesta Subseção serão sobre a Figura 34. A figura está dividida em duas partes e cada uma dessas partes representa uma das variações da arquitetura. A figura está subdividida da seguinte maneira:

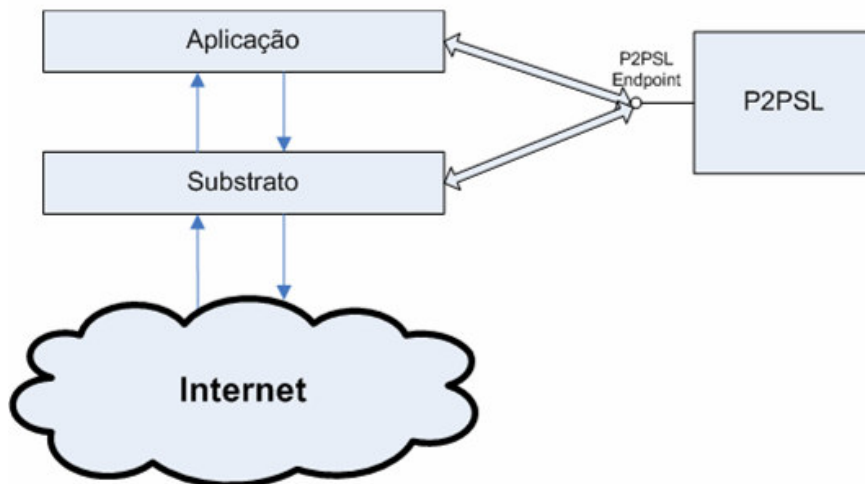
- a. Utilização da P2PSL em dois momentos usando API Java. Tanto a aplicação quanto o substrato devem gerenciar as mensagens de troca de requisitos e chaves da camada de segurança.
- b. Este modelo mostra a P2PSL como uma Arquitetura Orientada a Serviço onde qualquer parte de um sistema (como aplicação ou substrato) podem contactá-la através do *endpoint* da camada. Neste caso também existe a necessidade de gerência das mensagens da P2PSL pelas camadas de aplicação e substrato.

A utilização dessa arquitetura pode ser feita tanto em redes Estruturadas quanto em redes Não-Estruturadas satisfazendo os requisitos de segurança mesmo em ambientes

com roteamento de mensagens e, melhorando o modelo da Subseção 6.1.1, garante que sejam impostos requisitos de segurança para os dados do substrato de rede. Dessa forma, se um nó A tentar contactar um nó B através de um nó C, a camada de segurança irá aplicar os requisitos de segurança que A tem com B na camada de aplicação, porém irá aplicar os requisitos de segurança que A tem com C para o substrato. Quando a mensagem chegar em C, apenas os dados do substrato serão enviados para P2PSL para recuperação dos dados originais. De posse da mensagem, C irá aplicar os requisitos de segurança do substrato que C tem com B e então enviar a mensagem para B.



(a) Utilizando API Java



(b) Utilizando SOA

Figura 34: P2PSL satisfazendo requisitos da Aplicação e do Substrato de Rede

A Figura 35 ilustra o fluxo de dados em um nó da rede quando o mesmo deseja utilizar a P2PSL. Primeiro os dados da aplicação são submetidos sob os métodos da camada de segurança e são gerados os dados aplicando os módulos definidos no perfil correspondente. Depois disso, o substrato de rede contacta a P2PSL para proteger os seus dados gerando os dados seguros do substrato de rede.

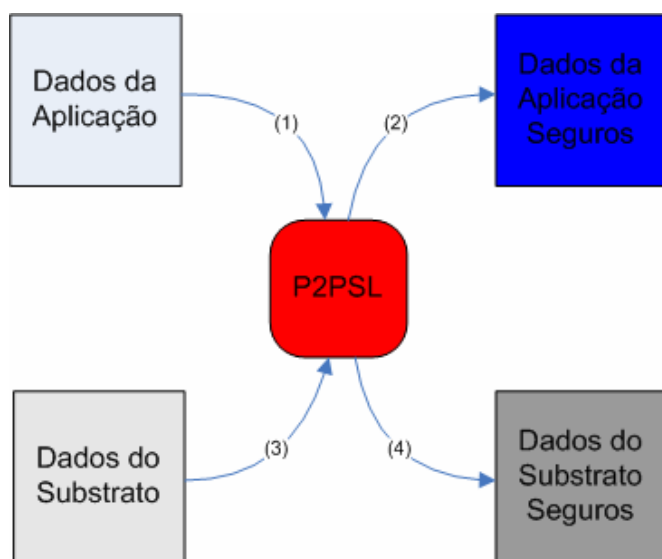


Figura 35: Fluxo de Dados

Uma vez que a aplicação e o substrato de rede tenham submetidos seus dados para P2PSL, pode-se montar o pacote final que irá ser enviado para um nó destino. A Figura 36 ilustra essa mensagem onde a mesma é composta pelos dados devolvidos pela P2PSL mais um identificador de nó origem (identificador do substrato) para que o nó que recebe os dados possa aplicar os módulos corretamente.

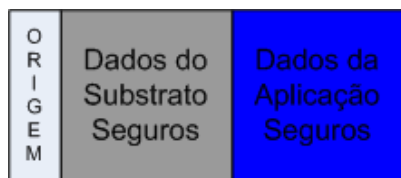


Figura 36: Mensagem Composta

Mensagens da P2PSL

Os modelos mostrados na Figura 34 utilizam uma única instância da P2PSL e fica a cargo do desenvolvedor gerenciar tanto na aplicação quanto no substrato as mensagens da camada de segurança. A troca de requisitos e chaves deve ser interceptada/requisitada pela camada que está se comunicando com a P2PSL e esta camada deve tratar de enviar os dados para o nó remoto para completar o protocolo descrito na Seção 4.5.

Nos dois modelos [(a) e (b)] descritos, existe a necessidade de uma API para gerenciamento das mensagens. No caso do modelo (b) isso será feito através de diferentes operações no *endpoint* da camada.

Mensagens da Aplicação

A aplicação pode optar pela utilização ou não na camada de segurança, pois a aplicação tem uma conexão direta com o substrato de rede. Caso a aplicação opte pela não utilização da camada de segurança, como o substrato não submete os dados da aplicação para P2PSL fazendo duplo encapsulamento, os dados da aplicação serão enviados pelo substrato para rede exatamente como foram criados pela aplicação. Caso a aplicação

opte pela utilização da P2PSL, o conteúdo da aplicação ficará protegido pelos requisitos do perfil correspondente.

Mensagens do Substrato

Existe a possibilidade do substrato contactar a rede sem a necessidade de aplicação de módulos de segurança em um caso em que o substrato “julgue” que não existe necessidade de aplicar requisitos de segurança. Mensagens de roteamento e manutenção do *overlay* podem atender aos requisitos de segurança sem ferir os requisitos da camada de aplicação, pois os dois serão independentes.

Características de Implementação

Essa arquitetura não necessita utilizar extensão de classes da P2PSL para desenvolvimento do sistema, pois a P2PSL pode ser usada como biblioteca. Por causa disso, as mensagens de uso exclusivo da camada de segurança precisam ser tratadas e gerenciadas tanto pela aplicação quanto pelo substrato de rede, pois a P2PSL não tem acesso direto a rede e necessita de uma forma de troca de requisitos.

Para o modelo (b), onde existe a utilização de SOA, é necessário implementação de protocolos SOAP tanto na aplicação quanto no substrato de rede tornando o desenvolvimento do sistema um pouco mais complexo, mas com isso a P2PSL pode ser utilizada com outras linguagens e plataformas.

Esse modelo trás grande flexibilidade, pois permite a escolha de utilização ou não da P2PSL em cada nó da rede. Dessa maneira, os nós que utilizam a camada de segurança podem optar por não utilizarem métodos da P2PSL caso comuniquem-se com nós que não possuem a camada de segurança eliminando o overhead imposto pela P2PSL.

6.2 Escalabilidade

Essa seção tem por objetivo propor soluções para sanar os problemas citados na Seção 5.2 melhorando a escalabilidade da P2PSL. Primeiramente será proposto um modelo que, teoricamente, é mais seguro que o modelo atual e aumentará a escalabilidade da camada de segurança, pois irá retirar um dos gargalos do sistema. Por fim, o modelo de troca de requisitos será levemente modificado visando aumentar a escalabilidade.

6.2.1 Troca de Chaves

Considerando os problemas descritos na Seção 5.2, o modelo proposto tenta aumentar a escalabilidade e reduzir problemas de segurança. Certificados digitais fornecem um nível de segurança bastante alto e juntamente com isso resolvem boa parte do problema da escalabilidade, pois como os certificados são sempre transmitidos diretamente entre nós sem a necessidade de envolver a autoridade certificadora, a autoridade não será sobrecarregada com requisições de chaves aumentando a escalabilidade do sistema como um todo.

Em termos de segurança, o uso de uma autoridade certificadora é mais confiável que o uso de autoridade de distribuição de chaves públicas, pois a autoridade certificadora não pode sofrer ataques na tentativa de modificar as chaves públicas dos usuários porque ela não mantém cópia das mesmas, pois o certificado será transmitido diretamente pelo nó que deseja identificar-se.

Além de aumentar a escalabilidade e fornecer um nível de segurança mais alto que o modelo anteriormente proposto, ainda deve-se tomar cuidado com alguns aspectos da utilização de certificados e das possibilidades de fraudar esse sistema, pois o problema

da inicialização do sistema (criação do certificado) também aparece nesse modelo caso um nó não tenha a chave pública da autoridade certificadora. A seguir serão apresentados alguns dos problemas conhecidos na utilização de certificados digitais.

Problemas Conhecidos na Utilização de Certificados

Deve-se ter em mente que para utilização de certificados digitais é preciso depositar confiança em uma entidade. Essa confiança pode ser traída tanto por má fé da autoridade quanto por suborno de algum funcionário que tem acesso a chave privada da autoridade. Além disso, pode-se ter várias autoridades certificadoras assinando certificados para clientes e com isso se tem outra pergunta: “Em quais autoridades devemos confiar?”. A sugestão é confiar em autoridades comprovadamente corretas em que muitas pessoas e entidades depositem plena confiança. Uma dessas autoridades é a Verisign (uma das mais utilizadas e que possui sua chave pública pré-instalada em vários navegadores).

Para os certificados mais simples, essas autoridades usam o e-mail como forma de garantir a autenticidade de um usuário, ou seja, o nome da pessoa e sua identificação está atrelada a uma conta de e-mail válida. Essa forma de certificação pode ser atacada caso um usuário tenha sua senha de e-mail exposta, pois com isso um atacante pode utilizar a conta de e-mail do atacado para criar um certificado em seu nome. Outra forma de obter esse certificado “forjado” é esperar um usuário de uma máquina se descuidar (deixando seu ambiente de trabalho desprotegido com caixa de e-mail aberta) e utilizar a conta de e-mail para gerar o certificado, fazer uma cópia e apagá-lo em seguida para evitar desconfiança.

Caso a chave privada de um usuário seja quebrada por métodos como força bruta, criptoanálise ou invasão de máquina é necessário que o usuário comunique a autoridade certificadora que teve sua chave privada descoberta. Com essa informação, a autoridade irá publicar em sua CRL que o certificado em questão é inválido. Porém, o tempo que essa informação pode levar para chegar a todos os interessados pode comprometer várias transações no sistema. A consulta excessiva da CRL da autoridade pode acabar transformando a mesma em um gargalo no sistema, por isso é importante verificar qual o *tradeoff* entre um tempo aceitável entre consultas à CRL e o tempo que uma chave pode ficar corrompida.

6.2.2 Troca de Requisitos

A solução para a baixa escalabilidade na troca de requisitos é simples. Ao invés de utilizar mensagens de *broadcast* no momento em que um nó entra na rede, a solução é fazer a requisição sobre demanda, ou seja, apenas quando existe a necessidade de contactar outro nó. Dificilmente, em uma rede P2P, será feita conexão com mais de 100 nós simultaneamente, ou seja, não existe a necessidade de contactar todos os nós da rede, pois muitos deles não são de interesse de um nó entrando no ambiente P2P.

Essa abordagem soluciona esse problema que estava presente na troca de requisitos. Tomando-se como exemplo uma ambiente como o KaZaA, a conexão de um nó se dá apenas com o Supernó sob o qual ele está subordinado e, posteriormente, com os nós que detêm um conteúdo de interesse para que o *download* seja feito em paralelo. No KaZaA, essa nova proposta faria com que a P2PSL trocasse requisitos apenas com o Supernó e os nós de interesse, não contactando outros nós na rede P2P.

6.3 Segurança

Esta seção tem por objetivo propor mudanças e melhorias para a camada de segurança P2PSL no que diz respeito às limitações citadas na Seção 5.3. Deve-se ter em mente que algumas soluções podem limitar a criação de alguns sistemas P2P se a utilização das mudanças for implementada sem flexibilidade no código. Deve existir a preocupação de sempre deixar a camada flexível o suficiente para que possa ser utilizada por qualquer proposta em qualquer momento.

6.3.1 Negação de Serviço

Ataques de nível de enlace não podem ser evitados ou controlados com a P2PSL, pois somente administradores de rede podem empregar políticas e filtros em roteadores e *switches* na tentativa de tornar ataques de esgotamento de capacidade de *link* menos efetivos. Juntamente com isso, o ataque de enviar muitas mensagens de buscas na tentativa de onerar a rede e os nós deve ser controlado no nível da aplicação, pois se a P2PSL limitar o recebimento de mensagens as aplicações que necessitam alta capacidade de transmissão serão prejudicadas.

Ataques de negação de serviço são difíceis de controlar em camada de aplicação, mas pode-se tentar diminuir sua eficácia quando agem apenas nessa camada. Para isso, a utilização de heurísticas de consumo de CPU e banda como a limitação do número de consultas pode ajudar a diminuir os efeitos do ataque, mas não irá garantir o controle de vários casos de ataque. Como esses controles devem ser feitos no nível da aplicação, não seria adequado incorporar esses comportamentos na P2PSL, pois a utilização, aparentemente, seria adequada para apenas alguns casos específicos.

6.3.2 Roteamento

Como esses ataques afetam exclusivamente o substrato de rede é necessário que o mesmo implemente mecanismos de segurança juntamente com a P2PSL para auxiliar a resolver os problemas, ou seja, é impraticável colocar toda a proteção na camada de segurança, pois o substrato precisa controlar e verificar mensagens. Para isso, a proposta de arquitetura da Subseção 6.1.3 é a mais adequada, pois facilita a obtenção de segurança nas mensagens enviadas pelo substrato.

As propostas descritas em [35] para detectar problemas de rotas incorretas e nós encaminhando mensagens para caminhos incorretos são de responsabilidade do substrato de rede e não podem ser auferidos apenas por uma camada que visa proteger os dados e fornecer garantias de maneira transparente como a P2PSL. Para implementar as soluções descritas é necessário inserir o teste de rotas antes de modificar a tabela e o controle de caminho percorrido por uma mensagem. O módulo de autenticidade pode ajudar o substrato a descobrir quem está enviando as mensagens maliciosas para que o substrato possa decidir o que fazer com o nó malicioso.

[43] propõe a utilização de uma Autoridade central que autentique os usuários que entram na rede gerando um certificado baseado no identificador. Isso faz com que usuários não possam escolher seu identificador, porém limita a escalabilidade do sistema. A P2PSL poderia ser usada com a proposta de certificados da Seção 6.3 de forma que o id fosse gerado baseado no *hash* de informações do certificado. Isso faria com que não fosse possível escolher um identificador qualquer na rede.

Com a restrição de geração de identificadores a partir do *hash* do certificado somado com o controle e investigação de rotas descrito acima faz com que nós incorretos possam ser mais facilmente identificados. Quando um nó identifica outro como malicioso, pode colocá-lo em uma lista negra e não contactá-lo novamente.

6.3.3 Autenticidade

Como citado na Subseção 5.3.3, garantir que uma chave pública depositada na Autoridade de Chave Pública não é uma tarefa simples, mas a proposta da Seção 6.2 auxilia nesse processo, pois a emissão de certificados por entidades de confiança é mais segura que o depósito de chave pública em uma Autoridade. Porém, isso não resolve totalmente o problema, pois qualquer pessoa que tem um e-mail é capaz de gerar um certificado e entrar na rede.

A simples adoção de certificados também não soluciona o problema de uma pessoa que tenha perdido a senha de seu e-mail e um terceiro não autorizado utilizado essa senha para gerar um certificado em seu nome. Porém, na tentativa de adotar uma medida de segurança mais restritiva onde resolva tanto esse problema quanto o problema de geração de certificados com e-mails falsos, pode-se configurar a camada de segurança para aceitar apenas certificados de alta confiabilidade como descrito no Capítulo 3 .

A utilização desta restrição garante qual pessoa física está enviando os dados digitais. Isso faz com que um atacante possa ser expulso para sempre do sistema, pois o mesmo não poderá obter outro certificado de alta confiabilidade, pois a Autoridade não permitirá. Isso também garante que uma pessoa possa ter acesso a apenas uma instância e apenas um identificador na rede. O problema dessa abordagem é o custo de implantação e a dificuldade de utilização, pois existe um custo fazer um certificado desse tipo. Além disso, existe necessidade presencial do interessado em fazer o certificado na Autoridade Certificadora sendo que isso pode ser problemático, pois se a intenção for implantar um sistema de alta utilização essa restrição fará com que não hajam adeptos.

6.3.4 Reputação e Confiança

O problema de reputação e confiança é abrangente e altamente dependente do usuário do sistema, pois existe a necessidade do usuário informar para o sistema de reputação de sua interação com outro usuário. Isso é necessário, pois sistemas que não necessitam de intervenção do usuário estão sujeitos ao ataque de poluidores. Muitas vezes também é necessário avaliar “manualmente” se uma interação com outro nó foi positiva ou não, pois pode ter havido tentativa de fraude.

Como existe essa necessidade de interação com o usuário e um módulo não pode ser implementado transparentemente, pode-se adotar uma das várias propostas de solução para esse problema já exploradas ([46],[50] e [51]). Esse trabalho não tem por objetivo propor um novo modelo de reputação e confiança.

6.3.5 Autorização

O módulo de controle de acesso RBAC oferecido pela P2PSL resolve o problema de autorização, mas, como citado na Subseção 5.3.6, os papéis estão associados aos nomes dos nós que podem ser gerados sem nenhuma restrição. Para resolver essa falha de segurança, o módulo RBAC pode substituir a utilização de nome de nó para obter acesso por assinatura digital.

Além disso, pode-se usar a proposta de controle de obtenção de identificadores na rede citada anteriormente ao invés de conferir a assinatura do nó. Para isso, o módulo RBAC precisa possuir uma ferramenta de auxílio externo que possa calcular o identificador do nó baseado em seu certificado. Assim, esse identificador pode ser usado nas listas de controle de acesso, pois toda vez que um usuário entrar na rede ele terá o mesmo identificador, pois o mesmo é baseado no *hash* do certificado.

6.3.6 Anonimidade e Negabilidade

Para resolver a questão de anonimidade utiliza-se roteamento anônimo, ou seja, não devem existir “pistas” no sistema que possam revelar o nó responsável por uma requisição ou quem deseja comunicar-se com quem. Na tentativa de alcançar esse requerimento, [44] descreve a criação de uma rede de roteadores cebola. Essa rede consegue fornecer anonimidade para um nó A que requisita recursos de um nó B, pois a mensagem passa por uma rede de roteadores que levam a mensagem de A para B sem que B saiba quem originou a mensagem.

Essa não é uma solução simples que poderia ser simplesmente incorporada em um módulo como os já existentes na P2PSL, assim seria adequado usar esse algoritmo na aplicação ou camada de rede utilizando os módulos da P2PSL para fazer a criptografia das mensagens. Dos itens descritos na Subseção 5.3.8, esse ambiente de roteamento anônimo atenderia a: anonimidade de consulta; anonimidade de leitor.

Atingir anonimidade de autor e publicador são características de nível de aplicação, pois é nela que estarão ou não contidas informações sobre conteúdo. Redes como Emula e KaZaA oferecem esse nível de anonimidade, pois um conteúdo publicado não necessita nem de autor e nem de publicador, mas apenas um lugar (repositório) que possa ser compartilhado. A anonimidade de servidor impediria que fosse possível descobrir quem está compartilhando o conteúdo.

Na tentativa de atingir a anonimidade do servidor, o esquema de roteadores cebola pode ser estendido para suportar esse requerimento. Para isso, basta que um nó A que faz uma busca, o faça através da rede de roteadores cebola, assim ninguém saberá quem está requisitando. Posteriormente, o nó, tome-se B, que tem o conteúdo desejado por A responde através da rede de roteadores cebola que utiliza a mesma rota para enviar o conteúdo para B. Dessa forma, A não sabe quem está lhe fornecendo o conteúdo e B não sabe quem está requisitando a informação.

Por fim, a anonimidade de documento que também é conhecida como negabilidade é uma característica de nível de aplicação, pois impede um nó de identificar o conteúdo que o mesmo armazena fazendo com que, teoricamente, o nó não pudesse ser responsabilizado pelo conteúdo que armazena. Freenet [45] é uma proposta com o intuito de prover anonimidade para inserção de conteúdo e negabilidade de dados.

6.3.7 Poluição de Conteúdo

O problema de poluição de arquivos tem uma abrangência muito grande no sistema, pois pode ocorrer a partir de qualquer entidade ou usuário. Não existe um modo de solucionar completamente esse problema em um sistema fracamente acoplado onde não se tem controle restrito e central. Para resolver parcialmente esse problema existem soluções que utilizam reputação ([46] e [47]), outras que utilizam ajuda explícita do usuário como o KaZaA e Emule (apagando conteúdo poluído e avisando outros usuários) e algumas que utilizam heurísticas e reconhecimento na tentativa de detectar arquivos poluídos ([48] e [49]).

Na tentativa de minimizar esse problema, um módulo auxiliar pode ser criado contendo algumas funções que auxiliem a tomada de decisão quando baixar ou não um arquivo. Esse módulo guarda os nós considerados amigos (fornecidos pela aplicação) e usa o julgamento desses nós sobre um arquivo indicando para a aplicação a condição do arquivo perante os nós de confiança. Então o usuário da aplicação poderá escolher se deseja realmente baixar o arquivo ou não. A tarefa de contactar os outros nós e coletar a informação seria totalmente realizada pelo módulo, deixando para o usuário/aplicação

informar para o módulo quais são os nós de confiança e qual o seu julgamento para um arquivo.

É importante salientar que, diferentemente dos outros módulos, o módulo proposto necessita de uma interação direta com a camada de aplicação e não pode ser usado transparentemente interceptando as mensagens de entrada e saída como os módulos já implementados na P2PSL fazem.

7 Implementação – P2PSL 2006

8 Conclusão

Esse trabalho investigou P2P e aspectos de segurança. Como se pode ver, ainda existem muitas questões em aberto e muito trabalho pra ser feito. Ou seja, publicações e dinheiro 😊

Bibliografia

- [1] MEL, H. X.; BAKER, Doris. **Cryptography Decrypted**. Addison-Wesley, 2001.
- [2] WILSON, Chuck. **GET SMART: The Emergence of Smart Cards in the United States and Their Pivotal Role in Internet Commerce**. Mullaney Publishing Group, 2001.
- [3] STALLINGS, William. **Cryptography and Network Security: Principles and Practice**. Prentice Hall, 1998.
- [4] BURNETT, Steve; PAINE, Stephen. **Criptografia e Segurança: O Guia Oficial RSA**. Editora Campus, 2002. (Tradução)
- [5] SCHNEIER, Bruce. **Applied Cryptography Second Edition: protocols, algorithms, and source code in C**. John Wiley & Sons, Inc., 1996.
- [6] KING, Christopher M.; DALTON, Curtis E.; OSMANOGLU, T. Ertem. **Security Architecture: Design, Deployment and Operations**. RSA Press, 2001.
- [7] DETSCH, André; GASPARY, Luciano Paschoal; BARCELLOS, Marinho Pilla; SANCEZ, Ricardo Nabinger. **Uma Abordagem para Incorporação Flexível de Aspectos de Segurança em Aplicações Peer-to-Peer**. Simpósio Brasileiro de Redes de Computadores, 2006, Curitiba-PR.
- [8] XIA, Haidong; BRUSTOLONI, José Carlos. **Hardening Web browsers against man-in-the-middle and eavesdropping attacks**. Proceedings of the 14th international conference on World Wide Web, 2005.
- [9] GONG, Li; ELLISON, Gary; DAGEFORDE, Mary. **Inside Java 2 Platform Security: Architecture, API Design, and Implementation, Second Edition**. Addison Wesley, 2003.
- [10] HOUSLEY, R.; FORD, W.; POLK, W.; SOLO, D. **RFC 2459 – Internet X.509 Public Key Infrastructure Certificate and CRL Profile**. IETF, 1999.
- [11] THEOTOKIS, S. A.; SPINELLIS, D. **A survey of peer-to-peer content distribution technologies**. *ACM Computing Surveys*, 36(4):335–371, 2004.
- [12] LUA, E. K.; Crowcroft, J.; PIAS, M., Sharma, R.; LIM, S. **A survey and comparison of peer-to-peer overlay network schemes**. *IEEE Communications Surveys & Tutorials*, 7(2):72–93, 2005.
- [13] Shirky, C. **What is p2p... and what isn't**. <http://www.oreillynet.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html>. O'Reilly, 2000.
- [14] ROCHA, J.; DOMINGUES, M.; CALLADO, A.; SOUTO, E.; SILVESTRE, G.; KAMIENSKI, C.; SADOK, D. **Peer-to-Peer: Computação Colaborativa na Internet**. Minicurso, Simpósio Brasileiro de Redes de Computadores, maio 2004.
- [15] SOUSA, Nuno Miguel Tavares de. **Peer-to-Peer Computing**. Universidade do Porto. http://gnomo.fe.up.pt/~eol/MEMBERS/nuno_sousa/old/ppc/artigo.html

- [16] MADRUGA, M.; BATISTA, Thais V.; GUEDES, Luiz A. **Uma Arquitetura P2P Baseada na Hierarquia do Endereçamento IP**. Em Simpósio Brasileiro de Redes de Computadores 2006.
- [17] **OpenNap: Open Source Napster Server**. <http://opennap.sourceforge.net/>, 2006.
- [18] RIPEANU, Matei; FOSTER, Ian; IAMNITCHI, Adriana. **Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design**. *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002.
- [19] LIANG, J.; KUMAR, R.; ROSS, K. W. **The kazaa overlay: A measurement study**. In *19th IEEE Annual Computer Communications Workshop (CCW 2004)*.
- [20] IZAL, M.; URVOY-KELLER, G.; BIRSACK, E. W.; FELBER, P. A.; HAMRA, A. All; GARCÉS-ERICE, L. **Dissecting BitTorrent: Five Months in a Torrent's Life Time**. In *Proceedings of the 5th Passive and Active Measurement Workshop*. Abril, 2004.
- [21] **DirectConnect**. <http://www.dcpp.net>, 2006.
- [22] **MSN Messenger**. <http://messenger.msn.com/>, 2006.
- [23] **ICQ.com**. <http://www.icq.com/>, 2006.
- [24] BASET, Salman A.; SCHULZRINNE, Henning. **An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol**. *Technical Report CUCS-039-04, Computer Science Department, Columbia University*. Setembro 2004.
- [25] ANDRADE, N.; CIRNE, W.; BRASILEIRO, F.; ROISENBERG, P. **OurGrid: An approach to easily assemble grids with equitable resource sharing**. In *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, Junho 2003.
- [26] SHEPLER, S.; CALLAGHAN, B.; ROBINSON, D.; THURLOW, R.; BEAME, C.; EISLER, M.; NOVECK, D. **RFC 3530 - Network File System (NFS) version 4 Protocol**. Abril 2003.
- [27] RHEA, Sean; EATON, Patrick; GEELS, Dennis; WEATHERSPOON, Hakim; ZHAO, Ben; KUBIATOWICZ, John. **Pond: the OceanStore prototype**. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*. 2003.
- [28] KNUTSSON, Bjorn; LU, Honhui; XU, Wei; HOPKINS, Brian. **Peer-to-Peer support for massively multiplayer games**. In *INFOCOM 2004*, volume 1, páginas 96-107, Março 2004.
- [29] Ratnasamy, S., Francis, P., Handley, M., Karp, R. **A scalable content-addressable network**. In *Proceedings of SIGCOMM 2001*.
- [30] STOICA, I.; MORRIS, R.; LIBEN-NOWELL, D.; KARGER, D. R.; KAASHOEK, M. F.; DABEK, F.; BALAKRISHNAN, H. **Chord: a scalable peer-to-peer lookup protocol for Internet applications**. In *IEEE/ACM Transactions on Networking*, 2003.
- [31] DETSCH, André. **Uma Arquitetura para Incorporação Modular de Aspectos de Segurança em Aplicações Peer-to-Peer**. Dissertação de Mestrado (Unisinos), 2005.
- [32] DETSCH, Andre; GASPARY, Luciano P.; BARCELLOS, Marinho P.; SANCHEZ, Ricardo N. **Flexible Security Configuration & Deployment in Peer-to-Peer Applications**. In *IFIP/IEEE Network Operations and Management Symposium (NOMS)*, 2006.

- [33] VERBEKE, Jerome; NADGIR, Neelakanth; RUETSCH, Greg; SHARAPOV, Ilya. **Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment.** *JXTA White Papers* in http://www.jxta.org/white_papers.html.
- [34] BARCELLOS, Marinho P.; GASPARY, Luciano P. **Fundamentos, Tecnologias e Tendências rumo a Redes P2P Seguras.** Jornadas de Atualização de Informática (JAI 2006), Vol 1, pg. 1-57.
- [35] SIT, E.; MORRIS, R. **Security Considerations for peer-to-peer distributed hash tables.** In *1st International Workshop on Peer-to-Peer Systems (IPTPS)*. 2002.
- [36] DOUCEUR, J. R. **The Sybil Attack.** In *1st International Workshop on Peer-to-Peer Systems (IPTPS)*. 2002.
- [37] TRAN, Huu; HITCHENS, Michael; VARADHARAJAN, Vijay; WATTERS, Paul. **A Trust based Access Control Framework for P2P File-Sharing Systems.** In *Proceedings of the 38th Hawaii International Conference on System Sciences*. 2005.
- [38] DINGLEDINE, R.; FREEDMAN, M.J.; MOLNAR, D. **The Free Haven Project: Distributed Anonymous Storage Service.** In *International Workshop on Designing Privacy Enhancing Technologies*, pg 67-95, 2001.
- [39] **Napster.** <http://www.napster.com>, 2006.
- [40] GUDGIN, Martin; HADLEY, Marc; MENDELSON, Noah; MOREAU, Jena-Jacques; NIELSEN, Henrik Frystyk. **SOAP Version 1.2.** W3C Recommendation 24 June 2003.
- [41] Sun Microsystems, Inc. **Project JXTA: An Open, Innovative Collaboration.** Draft 1.0 25 Abril, 2001.
- [42] CALLADO, Arthur; KAMIENSKI, Carlos; SADOK, Djamel H.; SOUTO, Eduardo; JÚNIOR, João; DOMINGUE, Marco A. O.; SILVESTRE, Guthemberg. **Peer-to-Peer: Computação Colaborativa na Internet.** Minicursos do XXII Simposio Brasileiro de Redes de Computadores (SBRC 2004).
- [43] CASTRO, Miguel; DRUSCHEL, Peter; GANESH, Ayalvadi; ROWSTRON, Antony; WALLACH, Dan S. **Secure routing for structured peer-to-peer overlay networks.** In *5th Usenix Symposium on Operation System Design and Implementation*. Dezembro 2002.
- [44] REED, Michael G.; SYVERSON, Paul F.; GOLDSCHLAG, David M. **Anonymous Connection and Onion Routing.** *IEEE Journal on Selected Areas in Communication*, Vol 16, No. 4, 1998.
- [45] CLARKE, Ian; SANDBERG, Oskar; WILEY, Brandon; HONG, Theodore W. **Freenet: A Distributed Anonymous Information Storage and Retrieval System.** *Lectures Notes in Computer Science*, 1999.
- [46] KAMVAR, S. D.; SCHLOSSER, M. T.; GARCIA-MOLINA H. **The EigenTrust Algorithm for Reputation Management in P2P Networks.** In *Proceedings of International WWW Conference*, 2003.
- [47] DUTTA, D.; GOEL, A.; GOVINDAN, R.; ZHANG, H. **The Design of a Distributed Rating Scheme for Peer-to-Peer Systems.** *Workshop on Economics Peer-to-Peer Systems*, 2003.
- [48] **Sig2dat tool for FastTrack Network.** <http://www.geocities.com/vlaibb/tools.html>, 2006.

- [49] **Audible Magic.** <http://www.audiblemagic.com>, 2006.
- [50] SRIVATSA, Mudhakar; XIONG, Li; LING, Liu. **TrustGuard: Countering Vulnerabilities in Reputation Management for Decentralized Overlay Networks.** *In Proceedings of the 14th International Conference on World Wide Web*, 2005.
- [51] SONG, S.; HWANG, K.; ZHOU, R.; KWOK, Y. K. **Trusted P2P Transactions with Fuzzy Reputation Aggregation.** *Internet Computing, IEEE*, Vol. 9, No. 6, 2005.
- [52] ZHUANG, Shelley Q.; ZHAO, Ben Y.; JOSEPH, Anthony D.; KATZ, Randy H.; KUBIATOWICZ, John. **Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination.** *The 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, 2001.